AccuSoft ®

# VisiQuest™

## Visual Programming Guide

# Table of Contents

Chapter 1: Imagine: VisiQuest . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
Command Line Options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
Overview of Graphical User Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
The VisiQuest Visual Programming Environment Workspace . . . . . . . . . . . . . 5
The VisiQuest Visual Programming Environment Menubar . . . . . . . . . . . . . . . 6
File Menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Edit Menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Workspace Menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Options Menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Control Menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Glyphs Menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Objects Menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Help Menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
The Workspace Command Bar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
The Console Window . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
Other VisiQuest GUI Features . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
Input and Output: Workspace Files and Workspace Objects . . . . . . . . . . . 7
Workspace Files VS Workspace Objects . . . . . . . . . . . . . . . . . . . . . . . . . . 8
Opening Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
Opening Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
Saving Files. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
Saving Workspace Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
The Visual Programming Workspace . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
The File Menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
New . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
Open File.... . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
Open Object... . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
Save . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
Save File... . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
Save Object . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
Close... . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
Exit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
The Edit Menu. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
Cut . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
Copy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
Paste . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
Show Clipboard . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
Duplicate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
Delete . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
Undo Delete . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
Select All . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

Unselect All . . . . 14
Find . . . . 14
The Workspace Menu. . . . 14
Run . . . . 14
Stop . . . . 14
Run Once / Run Continuous . . . . 14
Reset . . . . 15
Clear . . . . 15
Redraw. . . . 15
Information . . . . 15
Variables... . . . . 15
Configure Remote Hosts.... . . . . 15
The Options Menu . . . . 15
Adding a Toolbox Reference . . . . 15
Preferences.... . . . . 16
Hide Command Bar . . . . 16
Hide Console . . . . 16
Small Size . . . . 16
Clear Console . . . . 16
Reload Menus . . . . 16
The Control Menu . . . . 16
Create Procedure. . . . 16
Create Count Loop. . . . 16
Create While Loop . . . . 17
Create Expression . . . . 17
Create If/Else . . . . 17
Create Merge Paths. . . . 17
Create Switch. . . . 17
Create Trigger . . . . 17
The Glyphs Menu . . . . 17
The Help Menu . . . . 17
On VisiQuest . . . . 18
Activate Tooltips. . . . 18
License. . . . 18
The Workspace Command Bar . . . . 18

Introduction to Glyphs . . . . 19
Types of Glyphs . . . . 19
Standard Glyph Components . . . . 20
Input Data Connections . . . . 20
Output Data Connections. . . . 20
Pane Access . . . . 20
Run Button . . . . 20
Input Control Connections . . . . 21
Output Control Connection Node . . . . 21
Glyph Title . . . . 21
Open Workspace . . . . 21
Glyph Type Pixmap . . . . 21
Glyph Console Button as Error Indicator . . . . 21
Glyph Console Button as Info Indicator . . . . 21
Glyph ToolTips . . . . 21

Basic Glyph Operations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
Selecting a Glyph . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
Deselecting a Glyph . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
Moving A Glyph . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
Destroying A Glyph . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
Executing a Glyph . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
Changing the Glyph Title . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

Creating a Glyph . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
The Glyph Menus . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
Accelerated Routines List . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
Accelerated Glyph Finder List . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
Customizing Which Operators Are Accessible as Glyphs. . . . . . . . . . . . . . . . 26
Toolbox Contents VS. Operator Organization in VisiQuest Visual
Programming Environment . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

Input/Output: Glyph Connections . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
Data Connections . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
Yellow. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
Blue . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
Green . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
Control Connections . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
Manipulating Connections. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
Delete Connection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
Save Data to File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
Connection Options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

Operator Execution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
Executing (or Halting) the Visual Program In Its Entirety . . . . . . . . . . . . . . . 31
Run Mode . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
Executing (or Halting) a Single Glyph . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
Errors / Information Produced by Operator Execution . . . . . . . . . . . . . . . . . 32
Error Indication. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
Info Indication. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

Changing Operator Arguments (Using the Pane of a Glyph) . . . . . . . . . . 33
Text Selections . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
Filenames & File Browser . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
Other Selections: Logicals, Cycles, Lists, Etc. . . . . . . . . . . . . . . . . . . . . . . 34
Live Selections . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
Optional Selections . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
Toggles and Groups . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
The Run Button. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
The Help Button . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
The Close Button . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
The Options Menu. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
License . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
Composer... . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
Object Info.... . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
Runtime Logging . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
Process Debugging . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

GUI Editing and Save Changes . . . 36

Debugging . . . 37

General Debugging Information . . . 37

Running the Debugger . . . 38

Preferences . . . 38

Glyphs . . . 39

Connection Type . . . 39

Display Glyph Finder on Start-Up . . . 39

Redraw Glyph Connections . . . 39

Place Glyph . . . 40

Display Glyph . . . 40

Set Glyph Reporting . . . 40

Set Glyph Selection Color . . . 40

The Workspace . . . 41

Echo Execution . . . 41

Set Background Color . . . 41

Set Background Pixmap . . . 41

Data Transport Type . . . 42

The Canvas Grid . . . 42

Set Canvas Grid To . . . 42

Canvas Grid Size . . . 42

Grid Color . . . 42

The Command Bar . . . 43

Toolboxes . . . 44

Variables and Expressions . . . 44

Defining Variables . . . 45

Data Types . . . 46

Incrementing/Decrementing Variables . . . 46

Checking Values of Variables . . . 47

Evaluating Expressions Using Defined Variables . . . 47

Example: Comparisons . . . 47

Example: Use of Predefined Functions . . . 47

Example: Defining New Variables Depending on Previous Variables . . . 47

Example: Changing Values of Variables . . . 47

User-defined Functions . . . 47

Scope Of Variables . . . 48

Variables used by Loops and Expression Glyphs . . . 48

Using Variables with Filenames or Strings . . . 49

Deleting Variables . . . 50

Predefined Constants . . . 50

Predefined Functions . . . 51

If/Else Expressions . . . 52

User Defined Functions . . . 53

Logical Operators . . . 53

Arithmetic Operators . . . 54

Arithmetic Bit-wise Operators . . . 54

Arithmetic Assignment Operators . . . 55

String Operators . . . 55

Procedures . . . . 56
- Creating Procedures . . . . 56
- Opening/Closing Procedures . . . . 57
- Renaming Procedures . . . . 58
- Copying Procedures . . . . 58
- Nesting Procedures. . . . . 58

Control Structures . . . . 58
- Creation of a Control Glyph. . . . . 59
- Conditional Constructs . . . . 60
  - If/Else . . . . 60
  - Merge . . . . 62
  - Switch. . . . . 64
  - Trigger . . . . 65
  - Expression . . . . 66
- Looping Constructs . . . . 68
  - Creating A Loop Construct . . . . 68
  - Opening/Closing Loop Glyphs . . . . 68
  - Nesting Loops . . . . 68
  - Count Loop. . . . . 69
  - While Loop . . . . 71

Compiled Workspaces . . . . 74
- Workspace Compiler Types . . . . 75
  - The Default Workspace Compiler . . . . 76
  - The Script Workspace Compiler. . . . . 76
- Creation of Compiled Workspaces . . . . 76
  - Creating the Visual Program . . . . 77
  - Creating the Compiled Workspace. . . . . 77
  - Verifying Correctness . . . . 79
  - Modifying the Graphical User Interface . . . . 79
  - Modifying the Visual Network . . . . 82
  - Executing the Compiled Workspace from the Command Line. . . . . 83
  - Documentation And Maintenance . . . . 83

Data Transports . . . . 84
- Transport Buffering . . . . 85
  - File Buffering . . . . 85
  - Stream Buffering . . . . 85
  - Memory Buffering . . . . 86
- Directory for Creating Temporary Files . . . . 86

Distributed Computing . . . . 87
- Using the Remote Hosts Control Panel . . . . 87
- Dispatching a Glyph to a Remote Machine. . . . . 88
- Restrictions . . . . 89

Parallel Host Management . . . . 89
- Hosts . . . . 90
- Groups . . . . 90
- Daemons. . . . . 91
- Batch Queue . . . . 91

**Chapter 2: VisiQuest Operators . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1**

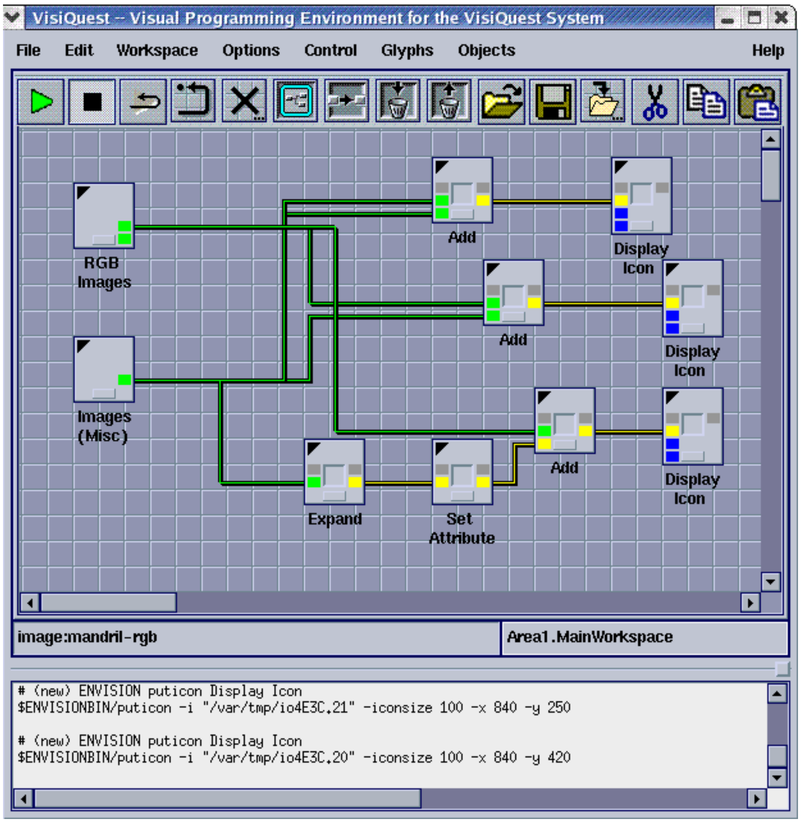
Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

Table of VisiQuest Operators . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

# Imagine: VisiQuest

## Introduction

VisiQuest Visual Programming Environment is used by the programmer to develop interactive programs visually by combining glyphs into a complete program. It is a graphically expressed, data flow visual programming environment that provides a visual programming environment within the VisiQuest system. Data flow is a "naturally visible" approach in which a visual program is described as a directed graph, where each node represents an operator or function and each directed arc represents a path over which data flows. The purpose in providing a visual programming environment interface to the programs included in VisiQuest is to increase the productivity of researchers and application developers. By providing a more natural environment which is similar to the block diagrams that are already familiar to practitioners in the field, the visual programming environment provides support to both novice and experienced programmers.



```
# (new) ENVISION puticon Display Icon
$ENVISIONBIN/puticon -i "/var/tmp/io4E3C.21" -iconsize 100 -x 840 -y 250

# (new) ENVISION puticon Display Icon
$ENVISIONBIN/puticon -i "/var/tmp/io4E3C.20" -iconsize 100 -x 840 -y 420
```

In VisiQuest Visual Programming Environment, the icons (called glyphs) typically represent programs from VisiQuest. However, given the VisiQuest software integration environment, they can also be used to represent non-VisiQuest programs that have been integrated into VisiQuest (see The Toolbox Programming Manual for information on creating a VisiQuest object and bringing the object into VisiQuest Visual Programming Environment). Each of the hundreds of stand-alone data processing and scientific visualization programs in VisiQuest can be represented in the VisiQuest Visual Programming Environment as glyphs. To create a VisiQuest Visual Programming Environment visual program, the user selects the desired programs (and control structures, as needed), places the corresponding glyphs on the VisiQuest Visual Programming Environment workspace, and connects these glyphs to indicate the flow of data from program to program, forming a network within a workspace. Such workspaces can be executed, saved, and restored to be used again or modified later. Workspaces may also be encapsulated into stand-alone applications with a very simplified graphical user interface so that they may be treated as independent VisiQuest Command Bars.

The visual hierarchy, iteration, flow control, and expression-based parameters make VisiQuest Visual Programming Environment a powerful simulation and prototyping system. VisiQuest Visual Programming Environment interprets the visual network dynamically to schedule glyphs and then dispatch them as processes. The VisiQuest Visual Programming Environment scheduler is event driven rather than data driven or demand driven. Glyphs are referred to as “coarse grained” because each glyph corresponds to an entire process, not a code segment or a sub-procedure. Once a glyph has been scheduled, the dispatcher is responsible for determining the data transport, the communication protocol, and the process execution mode. Communication protocol between VisiQuest Visual Programming Environment and the different glyphs can be as simple as just initiating process execution, or more complicated if glyph parameters must be continuously updated as the process executes continuously.

In UNIX environments, glyphs may be executed locally or remotely to efficiently utilize a heterogeneous network of computers. VisiQuest Visual Programming Environment utilizes a network execution daemon to negotiate the remote data transport and to spawn processes on remote machines. The visual programmer assigns operators to specific machines interactively; this may be done both to optimize execution speed and to fully utilize available hardware. Note that the remote machines to be utilized need not have a full VisiQuest installation, but must at least have a copy of the network execution daemon running in order to work with the remote transport mechanism.

Application specific domains, such as image processing and geometry visualization, typically process data as blocks. However, the domains of telecommunications and process control tend to process data as streams. VisiQuest Visual Programming Environment glyphs can process data in both blocks and streams.

The VisiQuest Visual Programming Environment extends the basic data flow paradigm to make it a more powerful application prototyping or simulation environment. Data and control-dependent program flow is provided by flow control glyphs such as if/else, while, count, and trigger. Visual subroutines, or procedures, are available to support the development of hierarchical data flow graphs. Variables may be set interactively by the user, or calculated at run time via mathematical expressions tied to data values or control variables.

## Command Line Options

```
% VisiQuest -usage

==========================================================================
Usage for VisiQuest:      "The visual programming language for VisiQuest"

% VisiQuest

  [-mini]       (flag)    Create A Miniaturized version of VisiQuest Visual
Programming Environment
  [-echoipc]    (flag)    Echo ipc port that VisiQuest Visual Programming
Environment is listening on
  [-tbmenus]    (boolean) Create VisiQuest Visual Programming Environment Pulldown
Toolbox Menus
                (default = true)
  [-commandbar] (boolean) Create VisiQuest Visual Programming Environment Command
Bar Menus
                (default = true)
  [-console]    (boolean) Create VisiQuest Visual Programming Environment Console
                (default = true)

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
    [-wksp] (infile) file for restoring VisiQuest workspace
            (default = {none})

    [Optional Mutually Inclusive Group - Specify all or none of the 2 options:]
      [-tb]    (string) the toolbox of the workspace object
               (default = {none})
      [-oname] (string) The workspace object name
               (default = {none})
  [-gui]        (flag)    run from GUI as defined in *.pane file
  [-V]          (flag)    gives the version number of the program
  [-U]          (flag)    gives brief usage
  [-usage]      (flag)    gives full usage
  [-P]          (flag)    interactive prompting for arguments
  [-x]          (integer) x value for GUI autoplacement
                (-1 <= x <= 1000, default = -1)
  [-y]          (integer) y value for GUI autoplacement
                (-1 <= y <= 1000, default = -1)
  [-A]          (outfile) Creates an answer file
                (default = {none})
  [-a]          (infile)  Uses an answer file
                (default = {none})
  [-ap]         (infile)  prints answer file values
                (default = {none})
=================================================================================
```

### [-wksp {workspace file}] OR [-tb {toolbox} -oname {object}]

When VisiQuest Visual Programming Environment is started without the [-wksp] and without the [-tb] and [-oname] arguments, it begins without a visual program (an empty workspace). When it is started with the [-wksp] option specifying a workspace file, VisiQuest Visual Programming Environment restores the workspace into the main workspace, and allows you to modify it.

When started with the [-tb] and [-oname] arguments, VisiQuest Visual Programming Environment will open the specified workspace object in the specified toolbox

### [-x {value}] [-y {value}]

The x and y arguments are usually specified as a pair. Together, they indicate the automatic placement of the GUI at the (x,y) location, where x and y are given as device (screen) coordinates. By default, x and y are both -1, indicating manual placement of the GUI.

### [-V]

This argument will print out the current version of the application.

*[-U]*

This argument will print out the command line argument structure for the application in less detail than is given here.

*[-P]*

This argument will enable interactive prompting for each of the command line arguments to the program.

*[-gui]*

This argument provides a graphical alternative to the interactive prompting mode described above; it brings up the GUI of the application as defined in its *.pane file. You may enter values for the start-up arguments as desired. When they are complete, click on the Run button to start the program.

*[-A {filename}]*

This argument will create an answer file containing the specified values of command line arguments for later use with the [-a] option. This CLUI answer file will be named as specified if a filename is given. If no filename is specified, the default filename "$HOME/visiquest.ans" will be used. Note: the "-A" argument is often used in conjunction with the "-P" argument; you are interactively prompted for arguments to the application, and those arguments are saved for later executions.

*[-a {filename}]*

This argument will use the answer file values for the command line arguments as they were saved earlier in a CLUI answer file. The specified CLUI answer file will be used if a filename is given. If no filename is specified, the default file "$HOME/visiquest.ans" will be used. No other command line arguments should be necessary when the "-a" argument is used.

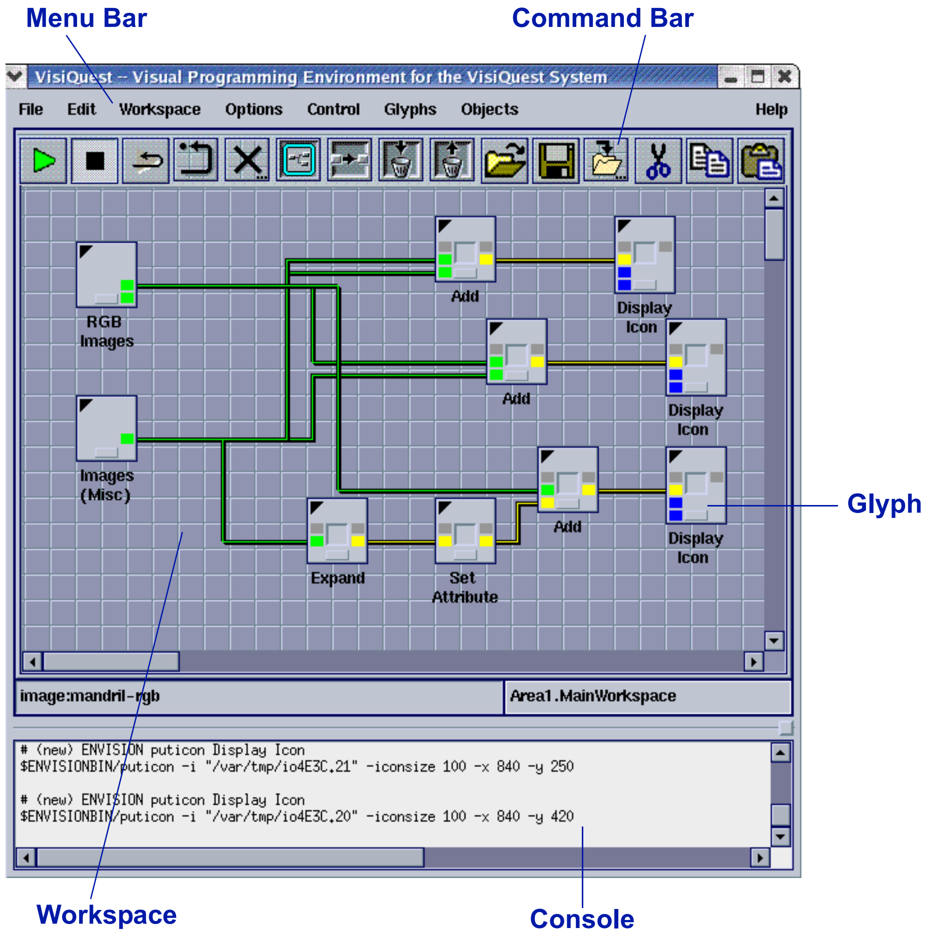
*[-ap {filename}]*

This argument will query the saved CLUI answer file for the command line argument values and echo them. The specified CLUI answer file will be queried if a filename is given. If no filename is specified, the default filename "$HOME/visiquest.ans" will be used.

## Overview of Graphical User Interface

The VisiQuest Visual Programming Environment graphical user interface consists of the following major components:

- The VisiQuest Visual Programming Environment Workspace
- The VisiQuest Visual Programming Environment Menubar
- The Workspace Command Bar
- The Console Window



```
# (new) ENVISION puticon Display Icon
$ENVISIONBIN/puticon -i "/var/tmp/io4E3C.21" -iconsize 100 -x 840 -y 250

# (new) ENVISION puticon Display Icon
$ENVISIONBIN/puticon -i "/var/tmp/io4E3C.20" -iconsize 100 -x 840 -y 420
```

## *The VisiQuest Visual Programming Environment Workspace*

The main portion of VisiQuest Visual Programming Environment is the VisiQuest Visual Programming Environment workspace. The workspace consists of a viewport containing a large canvas.  It is in this workspace that the visual programming network of glyphs is constructed.

To control whether or not grid lines appear, as well as the size of the grid and the grid color, select Options from the Tools menu and use the "Canvas" tab to set your preferences as desired.

The canvas is contained within a workspace and is many times larger than the size of the visible window.  You can control which part of the workspace is visible by moving the scroll bars on the left and bottom sides of the workspace frame or by enabling the Overview option in the View menu.

By default, VisiQuest Visual Programming Environment appears with a single workspace, but multiple workspaces can be created. To create a new workspace, select "New" from the File menu of the VisiQuest Visual Programming

Environment menubar. When multiple workspace areas are in use, tabs labeled "Area1", "Area2" and so on will appear below the VisiQuest menubar. The different workspaces can be brought to the foreground by clicking on the desired Area tabs. All actions are performed on the currently selected workspace canvas.

The workspace areas are managed as child windows in the main VisiQuest window. You can maximize them (as shown above), or have them displayed as independent, overlapping windows. When the workspaces displayed as child windows, you can have them automatically positioned, by using the Window menu Cascade and Tile items.

## *The VisiQuest Visual Programming Environment Menubar*

At the top of VisiQuest Visual Programming Environment is its menubar. This menubar contains a number of menus that provide access to all features of VisiQuest Visual Programming Environment as well as the VisiQuest operators. The menus available from the menubar are as follows:

### File Menu

The "File" menu contains items that allow you to create a new workspace area, open a VisiQuest workspace file, save the current network of glyphs as a VisiQuest workspace file, close the current workspace area, and exit VisiQuest.

### Edit Menu

The "Edit" menu contains utilities to edit to visual programming network, such as cut, paste, duplicate, delete, etc. Each of the actions on the "Edit" menu has a corresponding icon on the VisiQuest workspace command bar.

### Workspace Menu

The "Workspace" menu provides access to utilities for executing, halting, and manipulating visual programs in the VisiQuest workspace. It also provides access to the distributed computing and parallel computing capabilities.

### Options Menu

The "Options" menu provides various options for customizing the appearance and functionality of VisiQuest, including the Preferences subform.

### Control Menu

The "Control" menu allows the creation of special glyph types, including Procedure glyphs, while and Count Loop glyphs, and the various conditional glyphs. It also provides access to the VisiQuest workspace compiler.

### Glyphs Menu

The "Glyphs" menu provides access to the VisiQuest operators.

### Objects Menu

The "Objects" menu provides access to any Workspace Objects that may be available in the accessible VisiQuest toolboxes.

#### Help Menu

Standard help functions.

### The Workspace Command Bar

The Command Bar contains icons representing the most commonly used VisiQuest Visual Programming Environment commands.  The Command Bar is displayed just below the main VisiQuest Visual Programming Environment menu bar. The set of icons that appears is user configurable. You can control which icons you want to appear (including the entire set available). To select which icons appear in the command bar, select Preferences from the Options menu and use the "Command Bar" pane to specify your preferences as desired.

The workspace command bar can be hidden altogether by selecting the "Hide Command Bar" item on the Options menu, or by setting the command line option [-commandbar] to FALSE at startup.

### The Console Window

The console windowappears at the bottom of the VisiQuest Visual Programming Environment graphical user interface, and is divided from the VisiQuest Visual Programming Environment workspace with a sash. When a visual program is executed from VisiQuest Visual Programming Environment, the execution commands are echoed to the VisiQuest Visual Programming Environment console window (execution commands may also be echoed to a log file using the "Workspace" pane of the Preferences subform.

The console window can be hidden by selecting the "Hide Console" item on the Options menu, or by setting the command line option [-console] to FALSE at startup.

### Other VisiQuest GUI Features

In addition to the VisiQuest workspace, menubar, command bar, and console window, there is also a status window, a workspace name window, and a sash on the VisiQuest GUI. The status window, which appears directly below the workspace and to the left, is used by VisiQuest for reporting the status of various actions. The name window, to the right of the status window, indicates the name of the currently selected workspace.

The sash, below the status and name windows and above the console, is used to control how much of the VisiQuest GUI is devoted to the workspace, versus how much of the VisiQuest GUI is devoted to the console window. Click on the box at the far right of the sash to grab it and move it up or down.

## Input and Output: Workspace Files and Workspace Objects

The File menu on the VisiQuest Visual Programming Environment master form provides tools for:

- Opening and closing workspace areas
- Opening and saving VisiQuest workspace files

- Opening and saving VisiQuest workspace objects
- Exiting VisiQuest

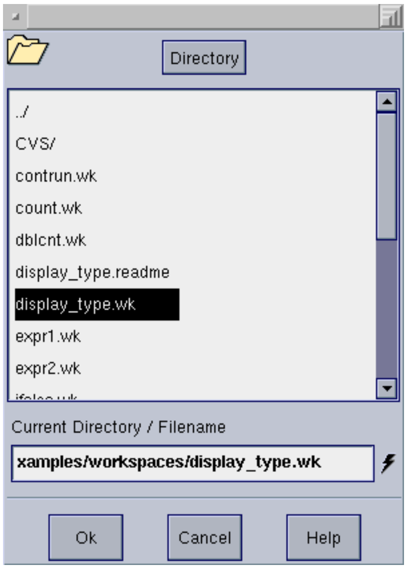
## Workspace Files VS Workspace Objects

VisiQuest Visual Programming Environment can save a visual program as a workspace file or as a workspace object. A workspace file is simply an ascii file containing lines that specify the glyphs, connections, variables, etc. that are in the visual program.

A workspace object is a software object that consists of the workspace file, a help file, and the software object database. Because they are accessible from the Object menus by category, subcategory and name, workspace objects are a nice way to "package" visual programs for other users or demos. Moreover, when a workspace object is restored, a special "help" icon will appear on the VisiQuest Visual Programming Environment command bar. You may click on this icon to access online help for the workspace.

## Opening Files

To open a saved VisiQuest Visual Programming Environment workspace file, select "Open File..." from the File menu. This displays the VisiQuest File Browser.



The Browser is used to locate and open a saved VisiQuest Visual Programming Environment workspace file. Use the mouse to select the file you want, or type the filename in the "Directory / Filename" text box at the bottom of the Browser, and click "OK." The saved workspace will be restored in the VisiQuest Visual Programming Environment main workspace. You will be prompted to delete any glyphs that were in the workspace before the "Open File" operation.

A shortcut to using the "Restore Workspace" selection of the "Files" subform is provided by the workspace command bar. Click on the "restore workspace" command button to display the file browser from which you may select a workspace to be automatically restored.

A saved workspace may be restored at startup time by using the [-wksp] option to VisiQuest Visual Programming Environment.

## Opening Objects

To open a workspace object, select "Open Object..." from the File menu. This brings up a Workspace Object browser and allows you to select a workspace object to open. The Workspace object browser is similar in appearance and operation to the Aliases browser. Initially, the list that appears will contain the categories of all workspace objects that are available. Selecting a category will cause the list to display all subcategories within the selected category. Selecting a subcategory will change the list so that it displays the names of all workspace objects in the subcategory. Finally, you may select a workspace object by name to be opened into the VisiQuest Visual Programming Environment workspace. The syntax used by the Workspace Object browser is "Category->Subcategory->Name"; you may also type in the desired workspace object using this syntax in the text box that appears near the bottom of the browser. The button at the top of the browser, labeled "Workspaces", is simply a queue that the browser is acting as a Workspace Object browser rather than a File browser or an Aliases browser; it will not respond to a mouse click.

A shortcut to using the "Open Object..." selection of the "Files" subform can be provided by the workspace command bar. Click on the "open object" command button (pictured above) to display the file browser from which you may select a workspace to be automatically restored. Note that by default, this icon is not displayed; you may use the Preferences subform to indicate that you want the "open object" command button to appear.

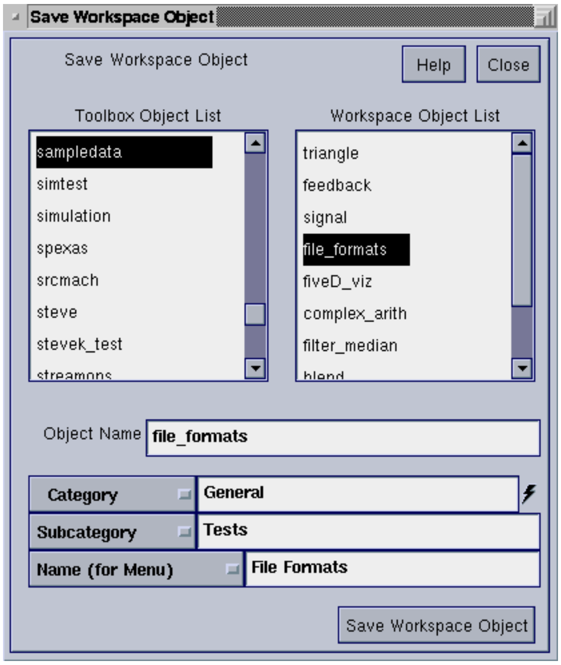
## Saving Files

To save a VisiQuest Visual Programming Environment workspace file, select "Save File..." from the File menu. This displays the VisiQuest File Browser. The browser is used to specify a new filename for the VisiQuest Visual Programming Environment workspace to be saved, or to select an existing file to be overwritten with the current VisiQuest Visual Programming Environment workspace.

A shortcut to using the "Save File..." selection of the "Files" subform can be provided by the workspace command bar. Click on the "save file" command button (pictured above) to display the file browser from which you may select a workspace to be automatically restored.

## *Saving Workspace Objects*

To save a VisiQuest Visual Programming Environment workspace object, select "Save Object..." from the File menu.



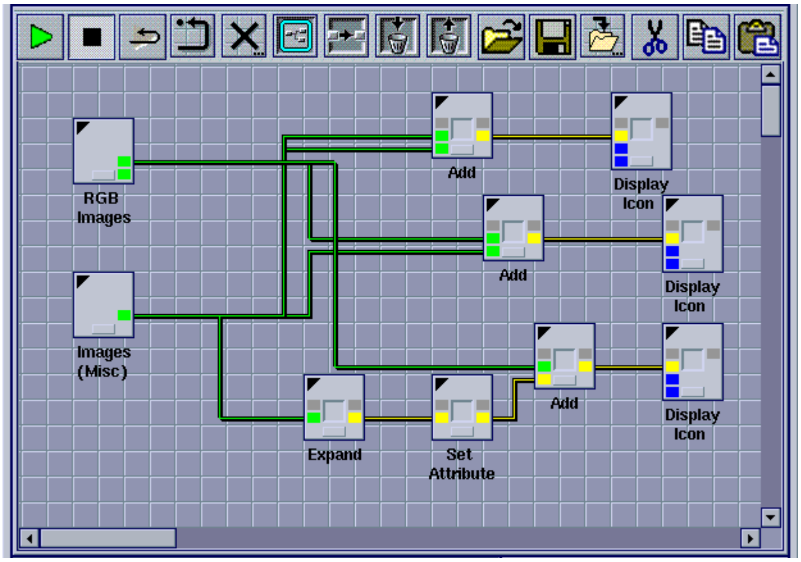
Object Name: file_formats
Category: General
Subcategory: Tests
Name (for Menu): File Formats

Then, select the toolbox in which the workspace object will reside. If there are any workspace objects that currently exist in that toolbox, those will appear in the Workspace Object List. You may over-write an old workspace object or create a new one. Specify the desired object name, as well as Category and Subcategory, and the name to appear in the Object Menus. Clicking on "Save Workspace Object" will cause the workspace object to be created. The workspace object can then be updated and maintained using Composer, just like any other software object. Remember to add text to the help file, so that the user will have access to meaningful online help describing your workspace.

A shortcut to using the "Save Object..." selection of the "Files" subform can be provided by the workspace command bar. Click on the "save object" command button (pictured above) to display the file browser from which you may select a workspace to be automatically restored. Note that by default, this icon is not displayed; you may use the Preferences subform to indicate that you want the "save object" command button to appear.

# The Visual Programming Workspace

The visual programming workspace is made up of a viewport containing a canvas with a grid. A visual program, made up of a network of glyphs, can be placed on the canvas, either by restoring a saved program, or by creating a new one. To the right and bottom of the canvas are scrollbars which can be used to control the portion of the grid that is visible within the viewport. Along the top of the workspace is a workspace command bar, containing buttons displaying icons.



These buttons provide shortcuts to a variety of visual programming operations that are also available from the main VisiQuest Visual Programming Environment menus, shown at the top of the VisiQuest Visual Programming Environment window.

There are a variety of functions supported by the VisiQuest Visual Programming Environment workspace. Procedure creation and loop construct creation are accessible from the Control menu. Editing capabilities are accessed from the Edit menu. File manipulation features are accessed through the File menu. The Workspace menu provides control for starting, stopping, resetting, and checking visual programs. The most commonly used functions are also accessible from buttons/icons on the workspace command bar. With only a few exceptions (where the concept simply does not apply), the workspace functions always operate on the currently selected glyphs.

Functions in VisiQuest Visual Programming Environment are available from several sources, such as the menus, the command bar, and keyboard accelerators. However, the most basic interface to VisiQuest Visual Programming Environment functionality is through the menus; while only a subset of operations may be available through the command bar, for example, all functionality is available through the menus. It makes sense, then, to discuss menus first.

The following sections explain the general organization of the VisiQuest Visual Programming Environment menus, with an explanation of the functionality provided by each item. Command bar counterparts to the menu items are provided where appropriate, as are keyboard accelerators. In those cases where a more detailed discussion is provided elsewhere, a reference to that discussion is given.

## The File Menu

The file menu provides tools to manipulate workspace files. Workspaces can be saved and restored using the utilities on this menu.

### New

New creates a new workspace area. By default, VisiQuest Visual Programming Environment appears with a single workspace area, but multiple workspace areas can be used. With multiple workspace areas, tabs labeled "Area1", "Area2" and soon will appear below the VisiQuest Visual Programming Environment menubar. The different workspaces can be brought to the foreground by clicking on the desired Area tabs. All actions are performed on the currently selected workspace.

A new workspace is displayed by default on top of the existing workspace. It may appear that the existing workspace has disappeared, but it is only hidden from view. It is possible to view other workspaces by clicking the Area buttons in the upper-left corner of the new workspace. Clicking the Area buttons allows you to toggle between multiple workspaces without closing and opening them.

Keyboard Accelerator: Ctrl-n

### Open File...

Clicking the Open item displays the VisiQuest Directory/Alias browser, which allows you to open a VisiQuest Visual Programming Environment workspace.

Keyboard Accelerator: Ctrl-o

### Open Object...

This item brings up a Workspace Object browser and allows you to select a workspace object to open.

Keyboard Accelerator: Ctrl-O

### Save

This writes out the current VisiQuest workspace to either a workspace file or to a workspace object, depending on what you did last. If "Open File" or "Save File" was used, then it saves the file as that filename. If "Open Object" or "Save Object" was used, then it allows you to save the workspace object.

Keyboard Accelerator: None

### Save File...

Save File allows the current workspace to be saved to a new filename.

Keyboard Accelerator: Ctrl-s

### Save Object

Save Object allows you to create a new workspace object.

Keyboard Accelerator: Ctrl-S

### Close...

When using multiple workspace areas, this closes the current workspace area. Note that the contents of the workspace area will not be saved. If you are using a single workspace area, "Close..." will produce an error message.

Keyboard Accelerator: Ctrl-w

### Exit

Select "Exit" to exit from VisiQuest Visual Programming Environment.

Keyboard Accelerator: Ctrl-q

## *The Edit Menu*

The Edit menu provides utilities for copying, deleting, and editing visual programs. It also provides tools for helping to locate glyphs.

### Cut

Cut removes a selected glyph (or glyphs) from the workspace and places it on the clipboard.

Keyboard Accelerator: Ctrl-x

### Copy

Copy makes a duplicate of the selected glyph and places it on the clipboard. The original is not removed from the workspace.

Keyboard Accelerator: Ctrl-c

### Paste

Paste copies glyphs from the clipboard and places them on the workspace.

Keyboard Accelerator: Ctrl-v

### Show Clipboard

Selecting "Show Clipboard" dispays the clipboard. Objects that are cut or pasted from the workspace using the relevant Edit menu commands are visible when in the clipboard. The clipboard can be used as a temporary repository for all or portions of a network that you wish to copy from one workspace and then paste into another within a single VisiQuest session.

### Duplicate

Duplicate makes a copy of a selected glyph or glyphs.

Keyboard Accelerator: Ctrl-d

### Delete

This item removes a selected glyph (or set of glyphs) without placing it on the clipboard.

Keyboard Accelerator: Ctrl-b

### Undo Delete

Undo Delete replaces a previously deleted glyph (or set of glyphs). It returns the most recently deleted item. If invoked a second time, it returns the item that was deleted before the last item, and so on through a series of previously deleted items.

Keyboard Accelerator: Ctrl-z

### Select All

This item selects all glyphs and connections in the entire workspace.

Keyboard Accelerator: Ctrl-a

### Unselect All

This item unselects any items in the workspace that are selected; if only one glyph is selected, it will be unselected; if many or all items are selected, they will all be unselected.

### Find

Launched by default when you start VisiQuest, this subform can be used to locate any of the glyphs available through the Glyph menus. The subform can be toggled back and forth between the Glyph Finder list and the Accelerated Routines list by selecting the labeled tabs at the top of the category display. See the sections on the Accelerated Routines List and the Accelerated Glyph Finder List for more details. You can change the default launch behavior in the Preferences subform (Options>Preferences>Glyphs).

## *The Workspace Menu*

The Workspace Menu provides access to utilities for executing, halting, and manipulating visual programs in the workspace.

### Run

Once glyphs have been selected and connected together in a network, the "Run" item may be used to execute the entire visual program automatically. Note that the Run icon on the workspace command bar, shown below, can also be used to execute all the glyphs in the workspace.

### Stop

Clicking the Stop button halts program execution. If the Run Mode is in continuous position, the scheduler will continue to run even though execution has been interrupted. If the Run button is clicked after the program has been halted, it will begin at the stage where it was stopped rather than starting from the beginning. To restart the program from the beginning, you must first click Reset (see below) and then Run.

### Run Once / Run Continuous

When Run Mode is is "Run Once" mode, VisiQuest executes the visual network only one time. Changes made in parameters to glyphs will not cause them to re-execute. When Run Mode is in "Run Continuous" mode, VisiQuest will execute the visual program continuously. As changes are made in parameters to glyphs, they will re-execute, and all downstream glyphs will also re-execute.

### Reset

Selecting "Reset" from the Workspace menu will touch all glyphs in the workspace so that a "Run" will reexecute ALL glyphs, whether or not they hav e been modified. Note that the "Reset" icon in the workspace command bar can also be used to reset all the glyphs in the workspace.

### Clear

Selecting "Clear" from the Workspace submenu will delete all glyphs in the workspace. If you accidentally clear the workspace without meaning to, the glyphs can be restored using the "Undo Delete" item on the Edit menu. Note that "Clear" icon on the workspace command bar can also be used to clear all glyphs in the workspace.

### Redraw

Selecting "Redraw" from the Workspace submenu will refresh all glyphs and the connections of the network in the workspace.

### Information

Selecting "Information" from the Workspace menu will calculate a few basic visual program statistics such as the number of glyphs and procedures in the visual program. Note that the "Information" icon on the workspace command bar can also be used to get information.

### Variables...

Selecting the "Variables" item displays the Variables subform. The subform can be used to define new variables, view the values of all variables, regardless of where they were defined, and watch them as they are incremented during program execution.

### Configure Remote Hosts...

Supporting distributed computing, VisiQuest allows programs to be executed on multiple machines. Selecting this menu item displays the Remote Host Control Panel, which is the user interface to VisiQuest's distributed computing feature.

## *The Options Menu*

The Options menu provides several items for customizing the appearance and functionality of VisiQuest.

### Adding a Toolbox Reference

This dialog box allows you to add a reference to another toolbox to your VisiQuest environment.  Note that this operation does not have anything to do with the currently selected toolbox; it is an independent operation. Adding a new toolbox reference will enable you to execute programs in that toolbox, modify its software objects using Craftsman and Composer, and access its operators in VisiQuest Visual Programming Environment.

Specify the path of the toolbox to which you wish to add a reference. Click "OK" to add the reference.  This will cause your main Toolboxes file to be updated with a line referencing the specified toolbox, and that toolbox will now be available in your VisiQuest environment.

### Preferences...

The "Preferences" subform allows workspace attributes to be changed. For example, the size and color of the grid, the texture and color of the canvas, or the visibility of the command bar.

### Hide Command Bar

This item makes the command bar disappear. After it has been selected, it toggles to "Show Command Bar" so that the command bar can be redisplayed.

### Hide Console

This item makes the console disappear. After it has been selected, it toggles to "Show Console" so that the console can be redisplayed.

### Small Size

Selecting this item reduces the overall size of the VisiQuest display. Reducing display size is handy if the program is executed on a system with a small monitor, say a laptop computer. Note that the size of the overall display is affected; icons, menu titles, glyphs, and other VisiQuest objects shrink along with the workspace. When the "Small Size" is selected, it toggles to "Large Size" to allow you to return to the original display size.

### Clear Console

Selecting this item erases all message appearing in the console window, just below the main workspace canvas.

### Reload Menus

Reloads the operators that are available as glyphs from the VisiQuest toolbox menus. This allows a newly created and installed program to be brought into a VisiQuest session without exiting and restarting VisiQuest.

## *The Control Menu*

The Control menu provides a series of utilities for creating special structures in the VisiQuest workspace.

### Create Procedure

Select this item to create a procedure. Creating procedures involves several steps.

### Create Count Loop

Select this item to create a count loop. Creating a count loop involves several steps.

### Create While Loop

Select this item to create a while loop. Creating a while loop involves several steps.

### Create Expression

Sets the value of a variable; used in conjunction with other control structures for purposes of variable initialization and/or modification.

### Create If/Else

Directs data flow to one path if a specified condition is met, to another path if the condition is not met.

### Create Merge Paths

Directs data flow from two separate paths to the same path, whether data arrives from the first path, the second path, or both; no condition is involved.

### Create Switch

Selects one of two inputs for use by the visual network, depending on the value of a conditional statement.

### Create Trigger

Delays execution of a glyph until data is made available by another glyph which is not otherwise connected to the dependent glyph. Control connections provide a simple way of specifying an order for process execution when one is not already dictated by the data flow, as is frequently the case in networks with a number of parallel paths.

## *The Glyphs Menu*

All of the VisiQuest rountines available as glyphs in VisiQuest are accessible through the Glyphs menu. The menu contents will change depending on the toolboxes that were actually loaded when VisiQuest was executed, but their general organization remains the same regardless. Clicking the Glyphs menu displays a drop-down list of categories. Selecting one of the categories displays a drop-down list of subcategories belonging to the parent category. Selecting a subcategory displays a drop-down list of glyph names. When a glyph name is selected, the outline of the glyph will appear in the VisiQuest workspace. Click the mouse in the workspace where you wish the glyph to be placed. The glyph will appear where you have placed it, the cursor will return to its arrow shape, and you may repeat the process to select other glyphs.

See the sections on the "Accelerated Routines List" and "Accelerated Glyph Finder List" for alternative methods of locating and placing glyphs on the workspace.

## *The Help Menu*

The Help menu provides access to information about VisiQuest. Information is organized under the following headings:

### On VisiQuest

Selecting this item displays information about VisiQuest.

### Activate Tooltips

This option serves as a reminder of the functionality of objects on the VisiQuest GUI. When activated, a brief explanatory phrase is displayed just below the cursor as the mouse passes over command bar icons, glyphs, connections, etc. Tooltips do not provide a detailed description of the functionality of these objects; rather, it supplies just enough additional information to help an experienced user remember the functional difference between similar looking items on the screen.

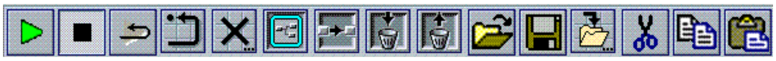
To prevent a constant flutter of descriptions from appearing as the mouse passes through the VisiQuest workspace, a slight delay occurs before the desciption is displayed; the mouse must remain over an object for two seconds before Tooltips display the message.

### License

Selecting this item provides information on VisiQuest licensing agreements for VisiQuest and other VisiQuest software.

## *The Workspace Command Bar*

The workspace command bar provides alternative interfaces to the most commonly used features of VisiQuest.

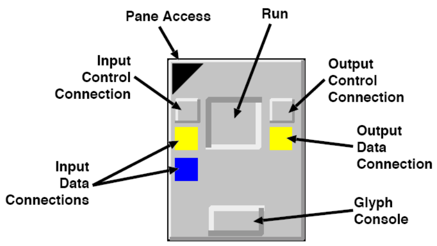


Many of the operations in VisiQuest that are available through Menus are also available from command bar icons. The same functionality that is invoked by selecting a menu item can be invoked by clicking a single icon. The intent is to make it easier to call those operations that are most commonly used. When VisiQuest is first executed, a default set of command bar icons is made available, but, by using the Workspace Preferences subform, the set can be edited to include more operations. Icons that are used infrequently can also be eliminated from the command bar.

Many of the icons that represent certain operations have been displayed along with the explanation of the corresponding operation, above. The command bar displayed above shows the general appearance of the icons in a group. If you are uncertain about the function of a particular icon in the command bar, place the mouse over it. A brief description of that icon’s functionality will display in the space betwen the console and the main VisiQuest workspace, close to the bottom of the VisiQuest window. Also, when the the Activate Tooltips option on the Help menu is turned on, a brief description displays just beneath the icon as the cursor passes over it.

# Introduction to Glyphs

A glyph is simply a visual representation of a program available from within the



**Glyph Title**

VisiQuest Visual Programming Environment.  Typically, these are the programs in one of the VisiQuest toolboxes you have installed at your site, but they can also be non-VisiQuest programs you have developed that have been given a VisiQuest pane interface (see The Toolbox Programming Manual for information on creating an VisiQuest object and bringing the object into VisiQuest Visual Programming Environment).

Each program may be run independently from the command line, or may be executed via VisiQuest Visual Programming Environment.

When accessed from VisiQuest Visual Programming Environment, the program itself is referred to as an operator; the icon that represents the operator in the VisiQuest Visual Programming Environment workspace is called a glyph.  As stated earlier, a visual program simply consists of a number of glyphs connected together in a network.

## *Types of Glyphs*

In fact, there are a number of different types of glyphs that are used in VisiQuest Visual Programming Environment.  A glyph that represents a VisiQuest program and allows the execution of an operator, as described above, is a standard glyph. The different glyphs available in VisiQuest Visual Programming Environment are as follows.

- Standard Glyphs

  A standard glyph represents an operator available from within VisiQuest Visual Programming Environment.

- Input Glyphs

  Input glyphs simply provide input for other glyphs. Usually, such glyphs provide a user interface for the specification of a data file.

- Procedure Glyphs

  Procedure glyphs represent a visual programming procedure consisting of a sub-network of glyphs.

- Control Glyphs

Control glyphs represent one of the conditional or looping constructs supported by VisiQuest Visual Programming Environment.

- Compiled Workspace Glyphs

  Compiled workspace glyphs represent a visual program that has been compiled into a separate binary or script. Internally, a compiled workspace is handled exactly like a standard glyph (e.g., an operator).

## Standard Glyph Components

A glyph has a number of components. Each component provides some sort of information about the glyph. In addition, many of the components are also buttons which you may use to perform an operation on the glyph. A summary of the various glyph components follows.

### Input Data Connections

Each glyph may have one or more input data connections. The input data connection node is represented by a colored square at the left edge of the glyph. When a data input connection is required, the square will appear in yellow; when it is optional, it will appear in blue. Some operators are provided for the express purpose of providing input for other operators; their glyphs will have no input data connections, as they take no input.

When there is data available to the operator from a previous glyph connected to the input data connection, the input data connection will change to green. Read this as, "data has been made available to the glyph at this input data connection."

### Output Data Connections

Each glyph may have one or more output data connections. The output data connection node is represented by a colored square at the right edge of the glyph. When a data output connection is required, the square will appear in yellow; when it is optional, it will appear in blue. Some operators are provided specifically to visualize data produced by other operators; their glyphs will have no output data connections, as they produce no output.

When data is made available by the operator to a subsequent glyph connected to the output data connection of the glyph, the data connection indicator will change to green. Read this as, "data has been made available by the glyph at this output data connection."

### Pane Access

Every glyph has a pane access button in the upper-left corner in the shape of a black triangle. This button is used to display the graphical user interface, or pane, of the operator. The pane is used to specify values for the arguments of the operator. These values correspond to command line arguments when the operator is run outside of VisiQuest Visual Programming Environment.

### Run Button

Most glyphs have a square run button in the center that is used to execute the operator represented by the glyph. The run button will glow red when the operator is executing.

### Input Control Connections

Input control connections are used to delay execution of the glyph until another glyph is executed.  This control is represented by the small, gray square above the input data connection(s).

### Output Control Connection Node

Output control connections are used to delay execution of another glyph until this glyph is executed.  This control is represented by the small, gray square above the output data connection(s).

### Glyph Title

Every glyph will display its title beneath it.  Typically this is the name of the operator that the glyph represents.  The glyph title can be changed by clicking on it, and then typing the desired title in the edit pop-up window.

### Open Workspace

Glyphs representing procedures and loop control structures have a special "Open Workspace" button which is used to open up the workspace associated with the procedure or the loop.  The open workspace button is the white triangle that appears in the upper-right corner of the glyph.

### Glyph Type Pixmap

Procedure glyphs and control glyphs have a special icon displayed in the middle to indicate the glyph type; this helps to differentiate them from standard glyphs. The glyph type pixmap is inside the run button square.

### Glyph Console Button as Error Indicator

The glyph console button at the bottom of the glyph normally appears in the same color as the background of the glyph.  However, if the operator encounters an error during execution and writes to standard error, the glyph console button will turn red.  Clicking on the glyph console button will display a message window containing the text that was written to standard error by the operator.  The glyph console button will appear in red until the glyph is reset.

### Glyph Console Button as Info Indicator

The glyph console button at the bottom of the glyph normally appears in the same color as the background of the glyph.  However, if the operator writes information to standard out during execution, the glyph console button will turn blue.  Clicking on the glyph console button will display a message window containing the text that was written to standard out by the operator. The glyph console button will appear in blue until the glyph is reset.

### Glyph ToolTips

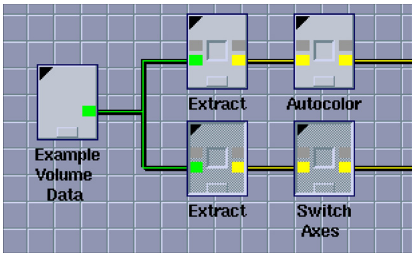
Glyphs have tooltips available; to activate tooltips, select "Activate ToolTips" from the Help menu of the VisiQuest Visual Programming Environment menubar. When tooltips are activated, move the mouse slowly over the glyph; an identifier for each component of the glyph will appear in a pop-up tooltips information box.

# *Basic Glyph Operations*

## Selecting a Glyph

Many of the workspace manipulation and editing capabilities in VisiQuest Visual Programming Environment work on the currently selected glyph(s). By selecting a glyph (or a set of glyphs), you are indicating to VisiQuest Visual Programming Environment which glyphs you want a particular operation to be applied. In general, if no glyphs are selected when such an operation is initiated, the operation will apply to no glyphs in the workspace. Selected glyphs will appear outlined in pastel blue or pastel green. The difference is that some functions, which operate on selected glyphs, need to know which of the selected glyphs is the “reference point” (for example, Tools/Format/Align) - the glyph outlined in green is the “reference point” (to which the other glyphs will be aligned).

You can select multiple glyphs by rubber banding (i.e., outlining with the mouse) a



box around a set of glyphs. To rubber band a box, click in the workspace at the upper left corner of the area containing the glyphs that you wish to select. Holding the mouse button down, drag the mouse to the lower right hand corner of the area containing the glyphs that you wish to select; a box will appear following the cursor as you move the mouse.

## Deselecting a Glyph

You can unselect a particular glyph by clicking in it and simultaneously holding the Shift button down. You can unselect all currently selected glyphs by clicking anywhere in the workspace canvas that a glyph does not appear. All glyphs in the workspace can be selected or unselected at once by choosing "Select All" or "Unselect All" from the Edit menu, or by using the "Select All" or "Unselect All" icons of the workspace Command Bar.

## Moving A Glyph

First select the glyph(s) to be moved. Then, hold down the left mouse button while dragging the glyph to the desired position. Releasing the mouse button will place the glyph(s) at the new location. When moving a set of selected glyphs, simply choose one glyph to drag to the new location; all the other selected glyphs will follow along, maintaining their relative position.

## Destroying A Glyph

To eliminate a glyph from the workspace, select it and then use the Delete item on the Edit menu. Multiple glyphs can be selected and deleted at the same time. You can also delete selected glyphs by pressing the Delete key on the keyboard.

If you destroy a glyph by mistake, you can recover the glyph by using the "Undo Delete" feature of the Edit menu, or the "Undo Delete" icon of the workspace Command Bar.

### Executing a Glyph

To execute the operator represented by a particular glyph individually, click on the run button of the glyph.

### Changing the Glyph Title

The readability of the visual program may be improved by changing the title of a glyph; this is particularly true of procedure glyphs. To change the title of a glyph, click on the current title that appears underneath the glyph; a prompt in which you can enter a new title will pop up. Enter the desired title and click "OK." The title that appears under the glyph will be immediately updated. While copies of the glyph will reflect the name change, note that the change is valid only for a single instance. That is, the name for the glyph in the Glyphs menu will not change.

## Creating a Glyph

Each VisiQuest program has an assigned category, subcategory, and icon name (also called the operator name). The use of the category/subcategory/name convention imposes a hierarchy on the VisiQuest Visual Programming Environment operators and makes the process of finding a particular operator from the hundreds of available operators a much easier task.

There are three ways to create a glyph for an existing operator in VisiQuest Visual Programming Environment. The first way uses the category/subcategory/name approach to finding the desired operator; the second way uses a combination of category/subcategory/name organization with alphabetization; the third way uses a combination of alphabetization and key word scanning. This section explains the three methods of glyph creation in VisiQuest Visual Programming Environment, and then goes on to describe how you can customize operators that are available as glyphs. Keep in mind that the operators available to you as glyphs in VisiQuest Visual Programming Environment will vary according to which VisiQuest toolboxes you have installed at your site, and which toolboxes you currently have installed in VisiQuest Visual Programming Environment.

### *The Glyph Menus*

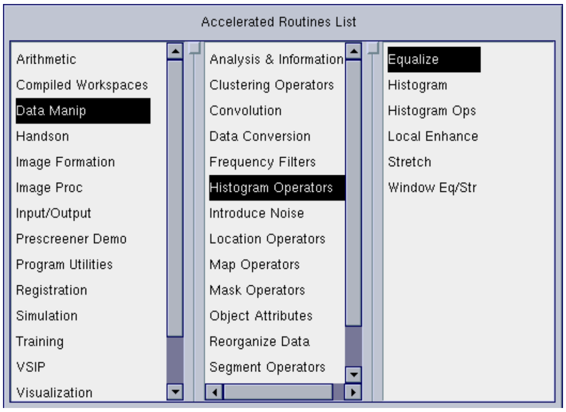
The Glyph Menus are used to select an operator and create a glyph according to category/subcategory/operator name.

The Glyph Menu provides the first method of glyph creation in VisiQuest Visual Programming Environment. The Glyph Menus contain a list of the operators contained in the enabled toolboxes arranged in their category/subcategory/name hierarchy. Selecting a category from Glyph Menu displays another walking menu of subcategories in that category. Selecting a subcategory displays a third walking menu of of operator names. Select the desired operator name, and then move the mouse into the VisiQuest workspace to place the new glyph. Note that the outline of the glyph will follow the cursor until you click the mouse in the workspace to place the glyph.

## Accelerated Routines List

Like the Toolbox Menu, the Accelerated Routines List is used to select an operator and create a glyph according to category/subcategory/operator name.

Click on the "Routines" subform button of the VisiQuest Visual Programming



Environment master form to display the Accelerated Routines List. This feature can be used as a short cut in selecting operators from which to create glyphs. Like the Toolbox Menu, the Accelerated Finder list also uses the category/ subcategory/name hierarchy to org anize operators. However, many users find the Accelerated routines list easier to use than the Toolbox Menu because the categories, subcategories, and operator names are all visible at the same time.

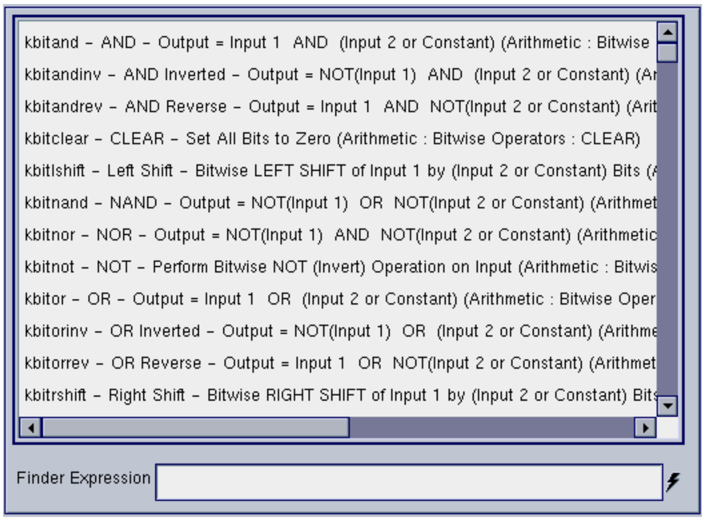
On the far left of the Accelerated Routines List is an alphabetized list of all the available categories. Clicking on a particular category will cause the middle list of alphabetized subcategories to be updated according to the contents of the selected category. Similarly, clicking on a particular subcategory in the middle list will cause the far right list of alphabetized operators to be updated according to the operators available in the selected subcategory.

To create a glyph, you may either double-click on an operator that appears in the list on the far right, or single click on an operator in the list and then click on the "Open" button that appears above the subcategory list.

After an operator has been selected from the list on the right, the "busy" cursor will appear while VisiQuest reads in the information associated with the selected operator. When VisiQuest is to create the glyph in the workspace, the cursor will change back to its original shape, and the outline of the glyph will follow the cursor until you click the mouse in the workspace to place the glyph.

## Accelerated Glyph Finder List

The Accelerated Finder List is used to select an operator by name; key words can



kbitand - AND - Output = Input 1 AND (Input 2 or Constant) (Arithmetic : Bitwise
kbitandinv - AND Inverted - Output = NOT(Input 1) AND (Input 2 or Constant) (Ar
kbitandrev - AND Reverse - Output = Input 1 AND NOT(Input 2 or Constant) (Arit
kbitclear - CLEAR - Set All Bits to Zero (Arithmetic : Bitwise Operators : CLEAR)
kbitlshift - Left Shift - Bitwise LEFT SHIFT of Input 1 by (Input 2 or Constant) Bits (A
kbitnand - NAND - Output = NOT(Input 1) OR NOT(Input 2 or Constant) (Arithmet
kbitnor - NOR - Output = NOT(Input 1) AND NOT(Input 2 or Constant) (Arithmetic
kbitnot - NOT - Perform Bitwise NOT (Invert) Operation on Input (Arithmetic : Bitwis
kbitor - OR - Output = Input 1 OR (Input 2 or Constant) (Arithmetic : Bitwise Oper
kbitorinv - OR Inverted - Output = NOT(Input 1) OR (Input 2 or Constant) (Arithme
kbitorrev - OR Reverse - Output = Input 1 OR NOT(Input 2 or Constant) (Arithmet
kbitrshift - Right Shift - Bitwise RIGHT SHIFT of Input 1 by (Input 2 or Constant) Bits

also be used to scan for a particular operator. Click on the "Finder" subform button of the VisiQuest master form to display the Accelerated Finder List. This feature provides a third method of glyph creation which is sometimes more convenient than either the Toolbox Menu or the Accelerated Routines List.

If you know the (binary) name of the program corresponding to the operator for which you wish to create a glyph, you can simply use the scrollbar on the left of the Finder list to scroll through all the available programs until the desired program name appears in the viewport (program names are always followed by their associated operator name).

If you do not know the name of the program in question, you may also use the Finder list to search for operators. You may type phrases, key words, or word segments in the Finder Expression parameter box that appears at the bottom of the Finder list. When you hit <cr> in the Finder Expression parameter box, the Finder list will scan all the available program names and their associated operator names, and update the list with only those having matches to the desired expression.

When the desired program and operator name appear in the finder list, you may create a glyph either by double-clicking on an operator that appears in the list, or by single clicking on an operator in the list and then clicking on the "Open" button that appears above the finder list.

After an operator has been selected from the list on the right, the "busy" cursor will appear while VisiQuest reads in the information associated with the selected operator. When VisiQuest is to create the glyph in the workspace, the cursor will change back to its original shape, and the outline of the glyph will follow the cursor until you click the mouse in the workspace to place the glyph.

## *Customizing Which Operators Are Accessible as Glyphs*

The operators available from the Glyph Menu, Accelerated Routines List, and Accelerated Finder List are dynamic. The complete set of available operators is determined by the toolboxes you have installed at your site. You may further control which of these toolboxes are available in the VisiQuest software development environment by using the "Add Toolbox Reference" and "Remove Toolbox Reference" features of craftsman. The subset of available toolboxes that are available to you in VisiQuest Visual Programming Environment may be further partitioned into enabled and disabled toolboxes using the Toolboxes pane of the VisiQuest Visual Programming Environment Preferences subform.

To enable and disable toolboxes in VisiQuest Visual Programming Environment, select the Preferences subform from the Options menu, and go to the Toolboxes pane. Enabled toolboxes appear in a list on the right, while disabled toolboxes appear in a list on the left. To disable a toolbox, select it from the Enabled Toolboxes list and click on the "Disable Toolbox" button. That toolbox will be moved to the list of disabled toolboxes.

Similarly, selecting a file from the Disabled Toolboxes list and clicking on "Enable Toolbox" will move the toolbox to the Enabled Toolboxes list. Click on the "Apply Changes" button at the bottom of the Preferences subform to enable and disable the toolboxes as specified.

To sav e the current configuration for the next VisiQuest Visual Programming Environment session, click on the "Save Preferences" button at the bottom of the Preferences subform.

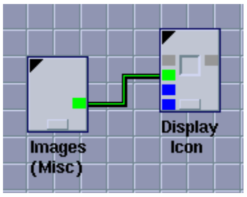
### Toolbox Contents VS. Operator Organization in VisiQuest Visual Programming Environment

There is no immediate indication in VisiQuest Visual Programming Environment of the toolbox in which an operator originated, since the Glyph menu, the Finder List, and the Accelerated Routines list all organize glyphs by category and subcategory rather than toolbox name. Furthermore, the categories and subcategories used by the Glyph Menu and the Accelerated Routines List span toolboxes; the appearance of a particular category does not necessarily imply that its contents will come from only one toolbox (although this is often the case). For example, the Retro and Image toolboxes both have operators in the "Image Proc" category and the "Transforms" subcategory.

Thus, if you disabled the Retro toolbox in VisiQuest Visual Programming Environment, the "Image Proc" category would still appear, since that category is still used by operators in the Image toolbox. Only if are no enabled toolboxes which use a particular category will that category will disappear from VisiQuest Visual Programming Environment.

# Input/Output: Glyph Connections

## *Data Connections*



To become part of a network, two glyphs are connected with a data connection. Here, the data connection between the two glyphs causes the output of the "Images (Misc)" operator to become the input of the "Display Image" operator.

Data connections are an integral part of the visual program, and are required for the program's construction.

Glyphs contain input and output data connection nodes, represented by colored squares located on the left and right sides of the glyph.  To create a data connection between two glyphs, do the following:

1. Position the mouse so that it is over a connector (data or control, input or output) on some glyph - the mouse cursor changes to a hand-with-pointing-finger
2. While holding the left mouse button down, move the mouse til it is over the appropriate connector on some other glyph (control-to-control, data-to-data, input-to-output).
3. Release the mouse button.

When a successful data flow connection is made, a connection line will be drawn between the two glyphs.

When two glyphs are connected with a data connection, it is implied that the output of the first will become the input of the second.  As such, the data connection represents data flow in the visual program. Color coding for data connection nodes is as follows:

### Yellow

If a data connection node is yellow, then its corresponding input/output parameter is a required argument for that operator. It must be connected in order to be executed. A yellow connection between two glyphs indicates that the upstream node does not have data available for the downstream glyph at its output data connection (ie, the glyph has yet to be executed and produce output).

### Blue

If a data connection node is blue, then its  corresponding input/output parameter is an optional argument for that operator. It may or may not be connected when the network is executed.

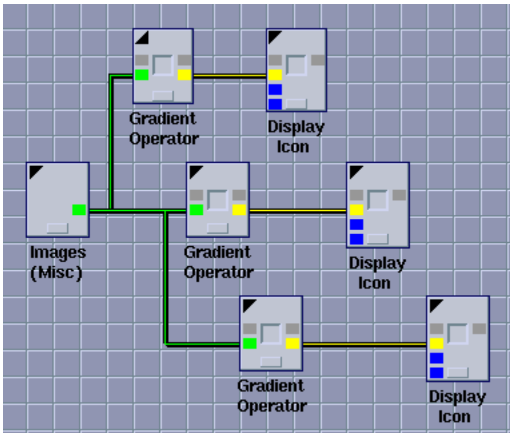
### Green

Both required and optional data connection nodes turn green when data is available at that point in the network (ie, the upstream glyph has data available for the downstream glyph). It will always be connected to other glyphs with data connections.

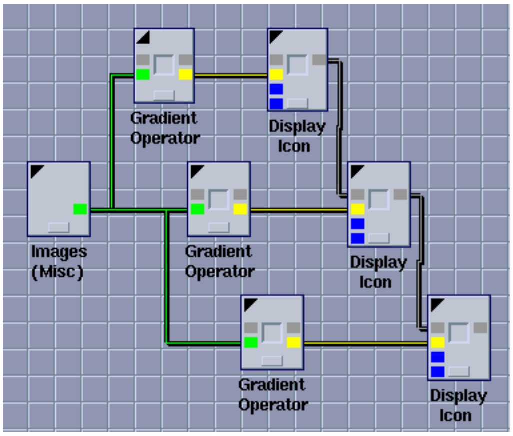
## Control Connections

A control connection is made between two glyphs to prevent the second glyph from executing until the first glyph has already done so. In addition to data connection nodes, glyphs contain a pair of input and output control connection nodes which are represented by dark grey squares at the upper left and right corners of the glyph. A control connection between two glyphs prevents the downstream glyph from starting execution until the upstream glyph is finished executing. While a visual program requires data connections between glyphs in order to form the network and to define where each process will obtain its data, control connections are not necessary as part of a fully operational network. They do, however, allow you to constrain the operation of a visual program and provide additional control over the order in which processes are executed.

Control connections simply cause the second, or controlled glyph, to "wait" on the execution of the operator represented by the first glyph.  Thus, control connections provide a simple way of specifying an order for process execution when one is not already dictated by the data flow, as is frequently the case in networks with a number of parallel paths.  Note that control connections can be created independently of data connections.  In other words, a glyph that does not feed data to a second glyph can still have a control connection to that glyph.



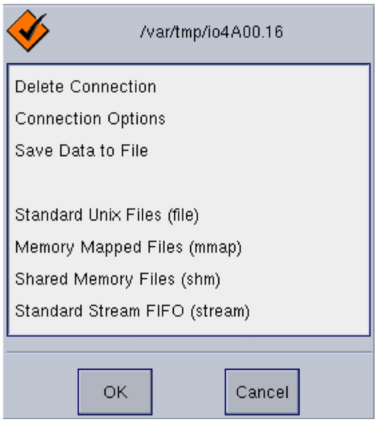
Without control connections, there is no way to predict the order in which the "Display Icon" operators will be executed.



With the control connections in place, the second "Display Image" operator will not be scheduled for execution until the first "Display Image" operator has displayed its image, and the user has closed the Display Image window.

## Manipulating Connections

Once a data flow or control connection has been made between two glyphs, it can be changed either by connecting that glyph to a different glyph or by deleting the connection altogether.


/var/tmp/io4A00.16
Delete Connection
Connection Options
Save Data to File
Standard Unix Files (file)
Memory Mapped Files (mmap)
Shared Memory Files (shm)
Standard Stream FIFO (stream)
OK
Cancel

Clicking the left mouse button on the connection between two glyphs will bring up a menu which you may use to delete the connection, save the file associated with that connection (for data flow connections only), or set connection options directly.

### Delete Connection

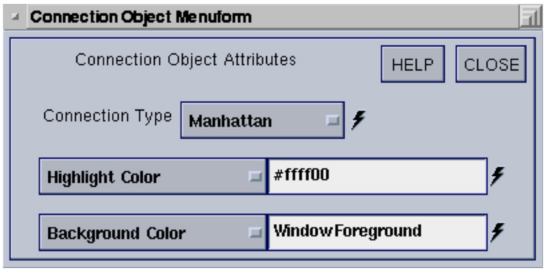
Choosing this selection from the menu will remove the connection.

### Save Data to File

Offered as an option with data flow connections only, this item will bring up a prompt where you can enter the filename in which to save a copy of the file associated with that connection. Note that this option can only be used with permanent data transport mechanisms, specifically shared memory, standard unix files, or memory mapped files.

### Connection Options

Selecting "Connection Options" from the Connection menu will bring up the menuform for the connection object. You can use the menuform to set attributes of the connection object, such as the line width of the connection, the connection type, the highlight color and background color used by the connection.


Connection Object Menuform

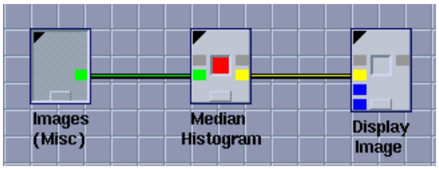
Connection Object Attributes

Connection Type: Manhattan

Highlight Color: #ffff00

Background Color: WindowForeground

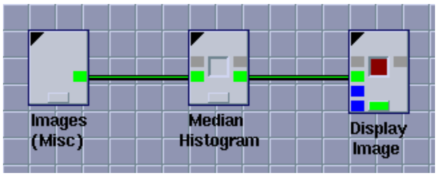
## Operator Execution



The Median Histogram glyph's "Run" button turns bright red while it is actively executing.

A single operator can be executed by clicking on the run button of its glyph. Alternatively, the entire visual program can be run by clicking on the "Run" button that in the workspace Command Bar (it’s a green triangle), or by selecting "Run" from the Workspace submenu.  Regardless of how the execution is initiated, both the "Run" button of the workspace Command Bar and the run button of the currently executing glyph(s) will be switched to the "on" position (the button on the glyph turns bright red) during execution.

If an operator is running but idle, as with continuously running glyphs that are waiting for new input and with xvroutines that have already been started up. Idle routines will switch to bright read again only when they receive new input, temporarily indicating a "non-idle" state until the new input has been processed and they are idle once more.



Once a visual program has been constructed, there are two ways in which you can execute the operators represented by the glyphs:

- You can run the entire visual program at once, where order of operator execution is determined by the data connections (and control connections, if any) of the network.
- You can execute individually one or more glyphs of the visual program "manually", by clicking their Run buttons in succession. The data and control connections in the workspace define the order in which the operators must be run. When executing a workspace manually, start with the Input Glyphs. For all other glyphs, you can run a glyph once all of its required input connections have turned green.

## *Executing (or Halting) the Visual Program In Its Entirety*

After a network is constructed, click on the run button of the workspace Command Bar or select "Run" from the Workspace menu to execute the entire visual program.  When you click the Run button, it will be disabled, and the Stop button will be enabled.

As each glyph in the visual network is executed, its individual run button will change from its "off" display to its "on" display (i.e., it glows red when running); thus, it is easy to see at a glance which glyph(s) are executing at any given time.

To halt the execution of a workspace, click on the Stop button.

## *Run Mode*

A VisiQuest workspace can be run in two different modes, described here and immediately below; the Run Mode button is used to control the scheduling mode. When the button is in single-iteration mode, it appears as above. In single-iteration mode, the workspace is scheduled and run exactly one time. Any changes to workspace parameters or input will not have an effect until the program is executed again by clicking the run button.

The Run Mode button is, by default, in responsive mode. Clicking the Run Mode button toggles it from responsive mode to single-iteration mode, and vice versa. In responsive mode, it appears as shown above. When a program is executed in responsive mode, the scheduler is always on, meaning that changes in parameters or input cause the program to rerun those segments that are affected by the changes as soon as the changes are detected by the scheduler. When the

Stop button is clicked, the program will cease to execute, but the scheduler will continue to run. If the Run button is clicked again, the program will continue execution from the state it was in when it was stopped rather than starting over from the beginning. If you wish to execute the program from the beginning after stopping it, click Reset before clicking Run.

When a program is executed, variables may be incremented, decremented, or changed in some fashion from their original values. Other program parameters, such as input files or calculated values, may also change during the course of execution. Clicking the reset button returns any altered values, parameters, or variables to their initial state. It also sets up the scheduler to re-examine the program from the point of view of its never having been executed, so once execution is invoked, the program is run in its entirety.

Because the network may contain several different paths, the scheduler will find a correct sequence for executing the glyphs and run them in that order. Once programs have been executed, if changes are made to the network, "Run" will re-execute only those programs that are affected by the change. Selecting "Reset" from the Workspace submenu (or clicking on the "Reset" button of the workspace command bar) will "touch" all glyphs in the workspace so that a subsequent "Run" will re-execute ALL glyphs, whether or not they have been modified.

## Executing (or Halting) a Single Glyph

Each executable glyph contains a run button which can be used to execute that operator individually. The run button is a square located in the center of the glyph. By default, this square appears gray, signalling that it is off. When the operator is being executed, the square turns red, signalling that it is on; when the operator is finished executing (or is halted) the square reverts to its gray, off display.

Clicking the run button causes the operator represented by the glyph to be executed. Clicking again on the run button halts execution of the operator.

**NOTE:** When executing glyphs "manually" as described here, it is important to follow the data flow of the visual network. This is because you cannot successfully execute a glyph unless all required input data are available as they would be if the entire visual program were executed automatically.

## Errors / Information Produced by Operator Execution

### Error Indication

If the execution of an operator fails, either because of an incorrect network connection or because of an error within the process itself, the glyph console button at the bottom of the glyph will turn red. Clicking the mouse on the glyph console button displays a message window containing the error output written to standard error by the operator.

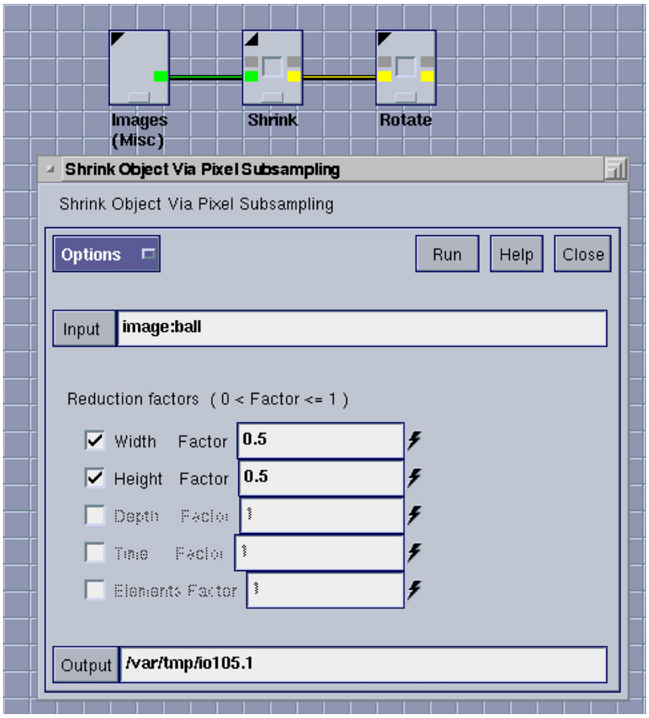
### Info Indication

The glyph console button at the bottom of the glyph will turn green when the operator produces information output during execution. Clicking the mouse on the glyph console button displays message window containing the text written by the operator to standard out.

## Changing Operator Arguments (Using the Pane of a Glyph)

Every program in VisiQuest has a pane, its graphical user interface that is used when the program is accessed as an operator in VisiQuest Visual Programming Environment. This GUI includes an item corresponding to each program argument so that you may use the GUI to specify the desired value for each.

When accessed via VisiQuest Visual Programming Environment, all input of arguments for an operator takes place on its pane. Thus, it is necessary to "open up the pane" for a glyph whenever parameters for the operator are to be changed. Open the pane for the glyph using the pane access button (the hand/document icon in the upper left corner of the glyph), and then change operator argument values as desired.



Shrink Object Via Pixel Subsampling

Input: image:ball

Reduction factors ( 0 < Factor <= 1 )

✓ Width Factor: 0.5

✓ Height Factor: 0.5

Depth Factor: 1

Time Factor: 1

Elements Factor: 1

Output: /var/tmp/io105.1

This is one of the most useful features of VisiQuest, because operator arguments can be changed while a workspace is running. After changing one or more of the arguments for a given operator, click the “Run” button in the Pane and that operator will be re-executed with the new arguments. Once it has completed, then all downstream operators will be re-executed, as per the data/control connections. This make it easy to “experiment” with a workspace.

Clicking on the pane access button of a glyph opens up its pane. Here, the pane for the "Shrink" operator is displayed; it consists of an input file selection, an output file selection, and 5 float selections for width, height, depth, time, and elements.

## Text Selections

Text selections, or text boxes as they are sometimes referred to, are provided for entering string, integer, float, and double values (and occasionally, filenames). Typically, these appear as a title, followed by narrow text parameter boxes in which the required information is entered by clicking the mouse pointer in the box and typing in a value; you can also switch from text box to text box by pressing the Tab key on the keyboard.

For bounded numerical values, scrollbars may be used to enter a value between predefined limits. Position the mouse cursor on the scrollbar, hold down the left button and move the mouse horizontally, releasing when it reflects the desired level.

Typing a specific value in the text parameter box always implies that the value entered will be a constant. In addition to entering constants, variables and expressions can also be used with integer, float, and double arguments.

## Filenames & File Browser

Names for input/output files may be typed directly. Alternatively, the file browser may be used to locate and specify a file. Clicking on either the Input File or the Output File button will bring up the file browser, which can be used to browse the directory structure and select a file. It also provides access to the aliases capability to select a file provided the SAMPLEDAT A toolbox has been installed. Directories may be accessed either by clicking on a subdirectory name (or ../ to back up) or by typing a pathname directly in the space at the bottom of the browser.

## Other Selections: Logicals, Cycles, Lists, Etc.

A variety of other types of selections are also found on panes; these include flags, logicals, cycles, pulldown lists, display lists, string lists, and so on.

In the case of a flag, a square button appears in front of a label; selecting the button implies a value of true for the flag, while leaving it unselected implies a value of false. For a logical, a title will be followed by a button. Clicking on the button will toggle back and forth between two labels indicating two boolean values. A cycle is similar to a logical, except that the button will indicate more than two values. A pulldown list looks similar to a logical or cycle, except that when you click on the button, a pulldown menu will appear, from which you may select a value. A displayed list is a window in which a number of items appear; the highlighted item indicates the current value of the selection. A stringlist looks like a string text selection, except that the title is really a button which you may use to automatically select one of a predefined set of strings; alternatively, you may enter your own string in the text parameter box. See the Toolbox Programmer's Manual for more details on the meaning and use of each type of GUI item.

## Live Selections

"Live" parameters are indicated by a stylized lightening bolt pixmap that appears to the right of the GUI selection box.  When a selection is "live," changing the value of the parameter will cause execution of the VisiQuest Visual Programming Environment network downstream from the glyph containing the modified pane.

**NOTE:** The Live Selection feature is not yet implemented on the Windows platform. On Windows, after changing the value of a selection in the pane, click the “Run” button at the top of the pane to have the operator re-executed with the new selection value. The glyphs downstream of the one whose selection value has changed will be re-executed automatically, if the workspace is running.

## Optional Selections

Optional parameters are prefaced by a small check box.  When selected, the box is checked, meaning the parameter value is to be used.  If the optional box is left unchecked, the parameter value is either to be ignored, or the default value used.

## Toggles and Groups

Relationships between optional selections are varied and sometimes complex. With a toggle, a set of selections having the same data type are grouped so that only one of the group may be selected at any time.  In contrast to toggles, groups do not have to be of the same data type.  A required mutually exclusive group expects exactly one member of the group to be selected at any time.  An optional mutually exclusive group expects only one or none of the group members to be selected at any time.  A mutually inclusive group requires all or none of the group members to be selected.  A loose group requires at least one group member to be selected.  Groups can be nested to one level.  For example, a mutually inclusive group could be nested inside a mutually exclusive group, requiring you to select at least one of A, B, or (C, D, and E).  A mutually exclusive group could be nested inside a mutually inclusive group, requiring that if you select A, you must also select B and one of (C, D, or E).  Using the grouping capability of the VisiQuest GUI, some panes can impose a complex relationship between variables.  Only a few panes use these more advanced capabilities.

## The Run Button

After all parameters on the pane have been set, the Run button allows execution of the program from the pane.  The program will execute using the argument values as specified on the pane.  Note that using the Run button on the pane is the same as using the run button on the glyph.

## The Help Button

The help button on the pane allows you to access the online help for the operator represented by the glyph.

## The Close Button

When you are finished with the pane, close it using the Close button; the glyph will be updated with the new values as specified in the pane.

## The Options Menu

Every pane has an Options submenu, which provides various operations regarding the program represented by the glyph pane.

### License

The License item displays the license associated with the program.

### Composer...

The composer item brings up the software object editor so that you can edit the program referenced by the glyph; in this way, VisiQuest integrates visual programming with the software development environment. Please see the documentation on Composer for more information about the software object editor.

### Object Info...

This selection will display an information window showing the current values of some of the program object attributes. Attributes displayed include the toolbox in which the program is installed, the name of the software object, the type ofsoftware object, and the category, subcategory, and icon name which is used with it in VisiQuest.

### Runtime Logging

When Runtime Logging is set to "on," a record is kept for how many times the program represented by the glyph executed. The duration of execution is also recorded, so the results are reported in terms of number of executions per second.

### Process Debugging

When Process Debugging is turned on, execution of the glyph will take place within the context of the debugger. A window with the debugging session will be displayed, and you can use the debugger on the program the same way as you would if it was executed from the command line.

The debug command used is specified with the VisiQuest_DEBUGGER environment variable.

See the following section for more details.

### GUI Editing and Save Changes

The pane of a program may be edited while the program is being accessed as an operator in VisiQuest. For example, the layout of the GUI items in the pane may be re-arranged, the titles of selections changed, and so on. The GUI Editing button puts the pane in "edit mode" so that you may make changes. Edit mode can also be toggled on and off by holding down the <shift> key and clicking the left mouse button in the pane. For more information about editing panes, please see the documentation on guise. After the pane of the operator is changed as desired, selecting "Save Changes" will save the changes to the GUI and re-generate the code of the operator.

**NOTE:** Be careful when editing panes! Changing the GUI of a glyph is not specific to a particular VisiQuest session; rather, it changes the GUI of the program itself, just as you would if you edited the associated program with the VisiQuest software development tools. Therefore, if you make a change, all other users at your site accessing that operator via VisiQuest will be affected by the change.

## *Debugging*

### General Debugging Information

New features and innovations in VisiQuest reflect an effort to establish it as a bona fide software development environment. One such feature, and a critical component for any serious software development environment, is the availability of an interactive debugger. Debuggers that are invoked from the command line are typically not available to visual programming environments like VisiQuest since there is no command line from which to call the debugging program. VisiQuest solves the problem by allowing the debugger to be called from the glyph pane.

As stated above, glyphs represent operators, software objects that perform specified data processing tasks. When the debugger is active, it attaches to the process generated by an operator when that operator is called in the course of running a visual program. Since an operator is a compiled software object, the debugger that is invoked by default is the one associated with the compiler used. For example, if the operator was compiled with gcc, the debugger will default to gdb, or if a Sun compiler was used, the default will be dbx.

However, choice of debugger is customizable, and can be set through an environment variable. The example below shows the syntax and arguments for setting the debugger on Solaris when the system is complied with Sun CC.

```
%setenv VisiQuest_DEBUGGER "xterm -e dbx %prog %pid"
```

Values that can be set in the argument string include:

**`%pid`** - active process id of process to be debugged

**`%prog`** - path to the program to be debugged

**`%fg`** - foreground color

**`%bg`** - background color

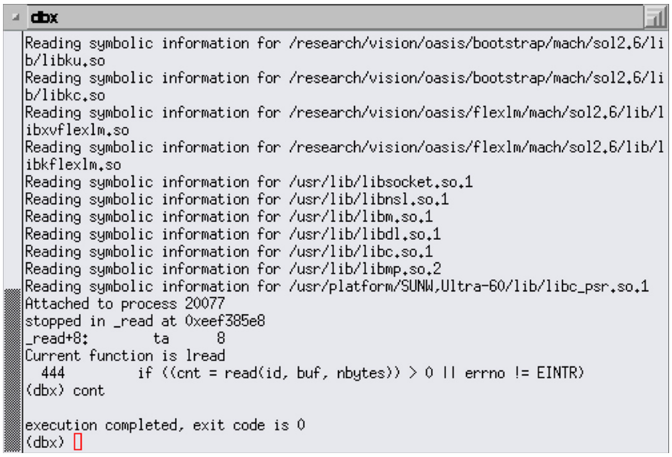
**`%g`** - geometry to be used

The debugger's attachment is local; the debugger will attach only to the process of the operator whose pane was used to invoke debugging in the first place, even if multiple instances of the operator occur in a visual program. If you wish the debugger to run for several operators, you must specify each case individually.

### Running the Debugger

To start the debugger, you must have a workspace open that you wish to debug. Identify the glyph that represents the operator you want to debug and opane the pane. Select "Process Debugging" from the Options menu to indicate that the operator in question is to run under the debugger.

Now run the program. When the flow of execution reaches the glyph for which you just ruend on Process Debugging, a window with the debugger running in it will display.



```
Reading symbolic information for /research/vision/oasis/bootstrap/mach/sol2.6/li
b/libku.so
Reading symbolic information for /research/vision/oasis/bootstrap/mach/sol2.6/li
b/libkc.so
Reading symbolic information for /research/vision/oasis/flexlm/mach/sol2.6/lib/l
ibxvflexlm.so
Reading symbolic information for /research/vision/oasis/flexlm/mach/sol2.6/lib/l
ibkflexlm.so
Reading symbolic information for /usr/lib/libsocket.so.1
Reading symbolic information for /usr/lib/libnsl.so.1
Reading symbolic information for /usr/lib/libm.so.1
Reading symbolic information for /usr/lib/libdl.so.1
Reading symbolic information for /usr/lib/libc.so.1
Reading symbolic information for /usr/lib/libmp.so.2
Reading symbolic information for /usr/platform/SUNW,Ultra-60/lib/libc_psr.so.1
Attached to process 20077
stopped in _read at 0xeef385e8
_read+8:        ta      8
Current function is lread
  444         if ((cnt = read(id, buf, nbytes)) > 0 || errno != EINTR)
(dbx) cont

execution completed, exit code is 0
(dbx) 
```

From this point, you can execute the regular set of dbx (or gdb, etc.) commands.

Notice that "Process Debugging" toggles from "<Off>" to "<On>" when process debugging is selected. Select it again to reverse the process and turn debugging off. Note that you must quit the debugging session to dismiss the debugging window. Also, be aware that a separate debugging window will display for each debugger invoked. This can be somewhat annoying if you happen to invike the debugger for an operator inside a while loop, or other iterative hierarchical strucure; a separate window will display for each loop iteration. You can avoid filling the screen with debugging windows by setting up the loop for debugging purposes to iterate only a couple of times. Return to the actual test condition when debugging is complete.

## Preferences

The Preferences subform, available from the Options menu on the VisiQuest menu bar, allows you to set parameters that control various aspects of the VisiQuest Visual Programming Environment. The Preferences subform consists of five separate subpanes that can be selected via the buttons at the top of the subform. The subpanes provide the ability to customize the VisiQuest GUI for the Glyphs, the Workspace, the Canvas Grid, and the Command Bar. In addition, toolboxes can be enabled and disabled from VisiQuest using the Preferences subform.

Preference settings take effect for the current VisiQuest session when the "Apply Changes" button is clicked. Note that changes indicated on subpanes that are not currently displayed are also invoked, so you can make changes on more than one subpane and click "Apply Changes" only one time to apply them all at once.

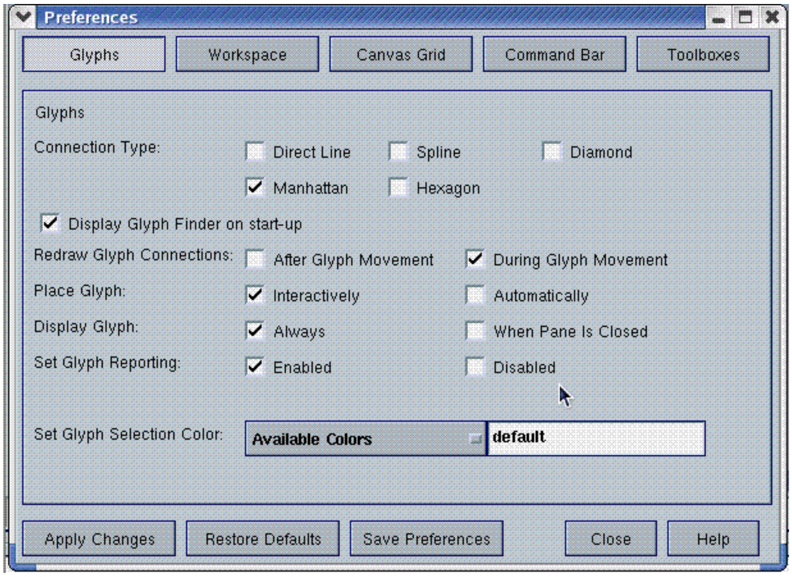
Clicking "Restore Defaults" returns the Preferences settings to those that were in effect when the current VisiQuest session was started.

"Save Preferences" saves the current Preferences settings to the $HOME/.kri/VQ/VisiQuestPrefs file. This file is automatically read in when VisiQuest is started, and the Preferences settings restored as specified.

"Close" dismisses the Preferences pane.

Each subpane and its attributes are described below.

## Glyphs



Glyphs

Connection Type: Direct Line [ ] Spline [ ] Diamond [ ] [✓ Manhattan] Hexagon [ ]

[✓ Display Glyph Finder on start-up]

Redraw Glyph Connections: After Glyph Movement [ ] [✓ During Glyph Movement]

Place Glyph: [✓ Interactively] Automatically [ ]

Display Glyph: [✓ Always] When Pane Is Closed [ ]

Set Glyph Reporting: [✓ Enabled] Disabled [ ]

Set Glyph Selection Color: Available Colors default

### Connection Type

This attribute allows you to control the type of connections that are used between glyphs. By default, the Manhattan (right angle cornered) connection is used. Also available are Direct Line (shortest route between two glyphs), Splines (curved connections), Hexagon (hexagon cornered), and Diamond (zigzag cornered) connections.

### Display Glyph Finder on Start-Up

By default, the Glyph Finder launches when you start up the VisiQuest Visual Programming Environment. If you do not want the Glyph Finder to launch on start-up, deselect this check box. You can also launch the Glyph Finder using Edit>Find.

### Redraw Glyph Connections

This refers to how the glyph connections are automatically adjusted whenever a glyph is moved. (Glyphs may be moved by placing the mouse cursor on the glyph, pressing the mouse button, dragging the glyph to the desired position, then releasing the button.) Using the value FALSE, the connections will be redrawn

AFTER the glyph is placed. Alternately, with "Connections" set to TRUE, VisiQuest will continuously update the connection WHILE the glyph is being moved. (Note, this may take longer than updating AFTER placement.)

### Place Glyph

When you create a glyph, it will be placed automatically if this attribute is set to FALSE. If you wish to place glyphs manually using the mouse, set this attribute to TRUE.

### Display Glyph

This attribute determines whether or not the glyph should be unmapped when its pane is opened. By default, the glyph is always displayed, whether or not its pane is opened.

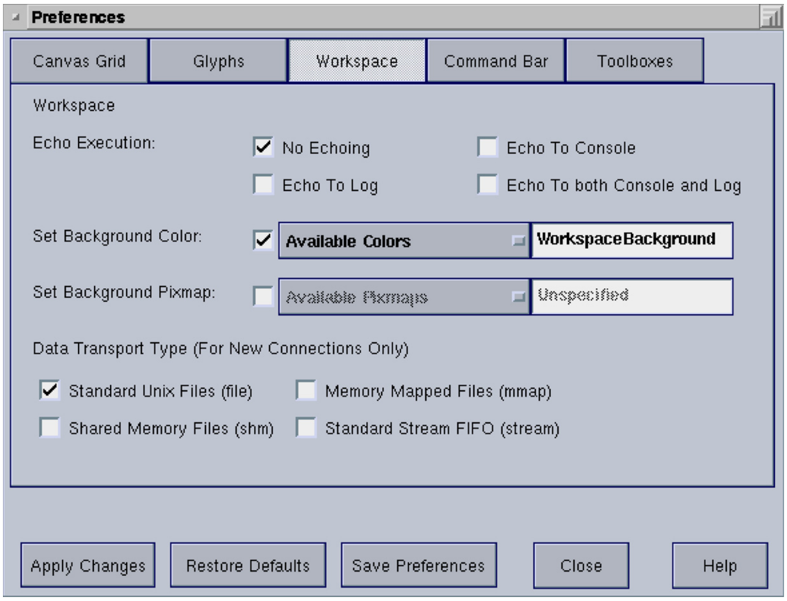
### Set Glyph Reporting

By default, this option is set to Enabled. When glyph reporting is enabled, if a process writes to stdout or stderr, the information icon will appear under the glyph; clicking on the information icon will display a window containing the text. If glyph reporting is set to Disabled, all text written to stdout or stderr will go to the console.

### Set Glyph Selection Color

This option will set a special highlight color for selected glyphs.

# The Workspace


Preferences

Canvas Grid | Glyphs | Workspace | Command Bar | Toolboxes

Workspace

Echo Execution: [✓ No Echoing] Echo To Console Echo To Log Echo To both Console and Log

Set Background Color: [✓] Available Colors: WorkspaceBackground

Set Background Pixmap: [ ] Available Pixmaps: Unspecified

Data Transport Type (For New Connections Only)

[✓ Standard Unix Files (file)] Memory Mapped Files (mmap) Shared Memory Files (shm) Standard Stream FIFO (stream)

## Echo Execution

This attribute causes the executing processes to be echoed to either the log, the console, or both, depending on which button under Echo Execution is selected. When process execution is echoed to the console, the output will appear in the console window below the VisiQuest workspace.

When process execution is echoed to the log, it will be written to the file which is specified by the VisiQuest_LOG environment variable. Values are as follows:

1. "No Echoing"

2. "Echo to Console"

3. "Echo to Log"

4. "Echo to both Console and Log"

## Set Background Color

The canvas background color can be set using this option. Enable the option by clicking toggle button. Click the "Available Colors" button to display a list of colors that can be used. Drag the mouse across the list and release it over the color you want to select. This option and "Set Background Pixmap" are mutually exclusive.

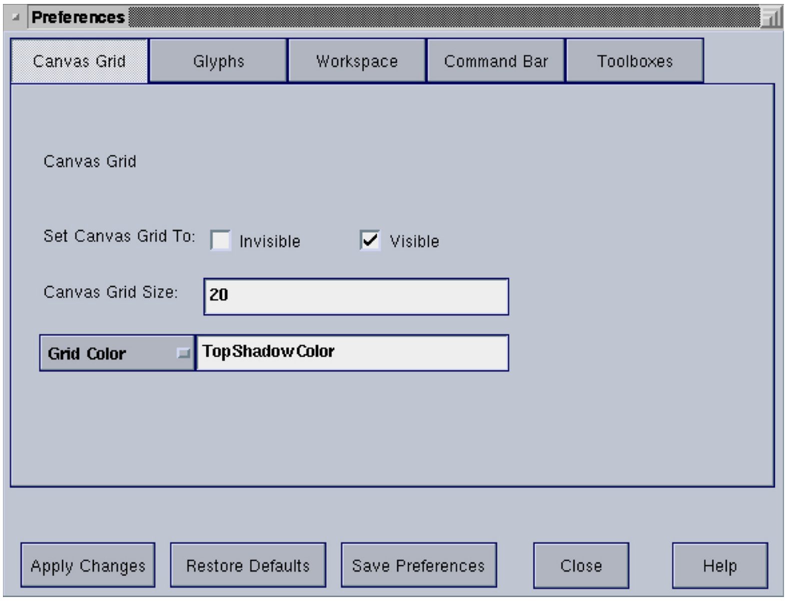
## Set Background Pixmap

The canvas background color can be set using this option. Enable the option by clicking toggle button. Click the "Available Pixmaps" button to display a list of pixmaps that can be used. Drag the mouse across the list and release it over the pixmap you want to select. This option and "Set Background Color" are mutually exclusive.

### Data Transport Type

This toggle allows you to set the data transport type that will be used for creation of all new data connections. It does not affect the data transport type of any existing data connections. For details on the concept of data transport and on the various data transports available in VisiQuest, see the Compiled Workspaces section.

## *The Canvas Grid*



Canvas Grid

Set Canvas Grid To: Invisible [✓ Visible]

Canvas Grid Size: 20

Grid Color: TopShadowColor

### Set Canvas Grid To

The canvas grid assists in the alignment of glyphs in the workspace. The grid may be made Visible or Invisible by selecting the corresponding button in the Preferences pane. Glyphs will snap to the upper-left corner of the nearest grid whether or not the grid is visible.

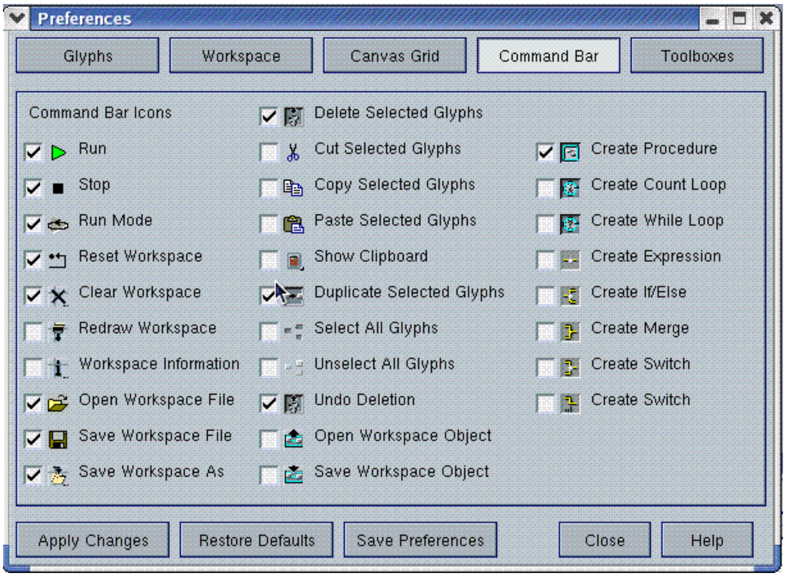
### Canvas Grid Size

If the grid is set to Visible, you may set the grid size (in number of pixels). A grid size of 1, with the "Set Canvas Grid To" attribute set to Not Visible, is effectively requesting "no grid."

### Grid Color

if the grid is set to Visible, grid color can be set using the Grid Color option. Click the labeled button to display a list of available colors. Drag the mouse over the color list and release it on the preferred color. You can also type a color in the text box, but be aware that some colors may not be available or the names you for them use may not be recognized.

# The Command Bar

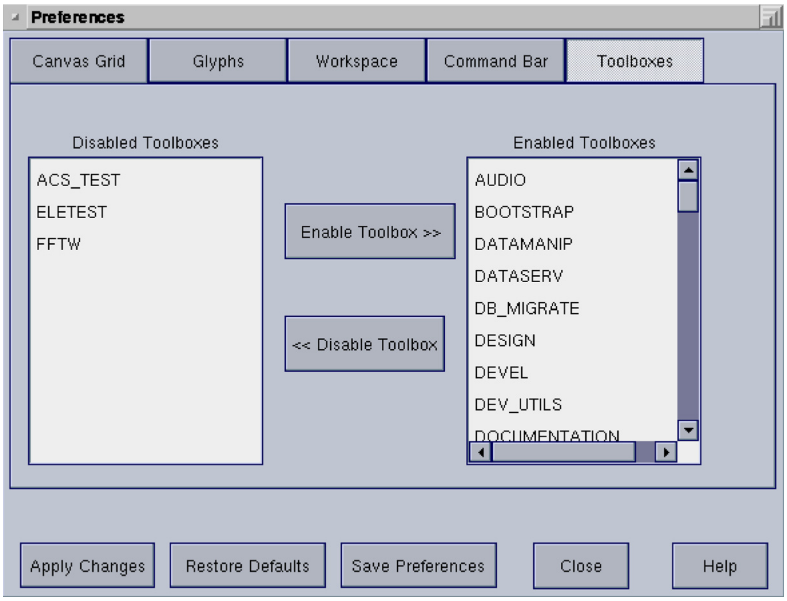


Command Bar Icons

- [✓] Run
- [✓] Stop
- [✓] Run Mode
- [✓] Reset Workspace
- [✓] Clear Workspace
- [ ] Redraw Workspace
- [ ] Workspace Information
- [✓] Open Workspace File
- [✓] Save Workspace File
- [✓] Save Workspace As
- [✓] Delete Selected Glyphs
- [ ] Cut Selected Glyphs
- [ ] Copy Selected Glyphs
- [ ] Paste Selected Glyphs
- [ ] Show Clipboard
- [✓] Duplicate Selected Glyphs
- [ ] Select All Glyphs
- [ ] Unselect All Glyphs
- [✓] Undo Deletion
- [ ] Open Workspace Object
- [ ] Save Workspace Object
- [✓] Create Procedure
- [ ] Create Count Loop
- [ ] Create While Loop
- [ ] Create Expression
- [ ] Create If/Else
- [ ] Create Merge
- [ ] Create Switch
- [ ] Create Switch

The group of command bar icons can be customized using the options on this pane. Any single icon can be displayed or hidden depending on whether it is selected or not on this pane. Command bar icons provide a swift method of invoking VisiQuest commands without having to use the menus. If there are commands that you use frequently, it is convenient to display the icon for that command on the command bar so that you can access it easily. Those commands that you use infrequently can be hidden and accessed via the menus. A default set of icons appears on the command bar when VisiQuest first displays, but your preferences can be saved so that they display automatically in future sessions. The entire command bar can be hidden by selecting "Hide Command Bar" from the Options menu. It can be redisplayed by selecting the counterpart command, "Show Command Bar" from the same menu.

## *Toolboxes*



Disabled Toolboxes: ACS_TEST, ELETEST, FFTW

Enabled Toolboxes: AUDIO, BOOTSTRAP, DATAMANIP, DATASERV, DB_MIGRATE, DESIGN, DEVEL, DEV_UTILS, DOCUMENTATION

When a toolbox is enabled in VisiQuest, all programs in that toolbox that have their "Install in VisiQuest" attribute set to TRUE will appear as operators accessable from the Glyph menus, the Accelerated Routines List, and the Finder List. When a toolbox is disabled, its operators will not appear.

Enabled toolboxes appear in a list on the right, while disabled toolboxes appear in a list on the left. To disable a toolbox, select it from the Enabled Toolboxes list and click on the "Disable Toolbox" button. That toolbox will be moved to the list of disabled toolboxes. Similarly, selecting a file from the Disabled Toolboxes list and clicking on "Enable Toolbox" will move the toolbox to the Enabled Toolboxes list.

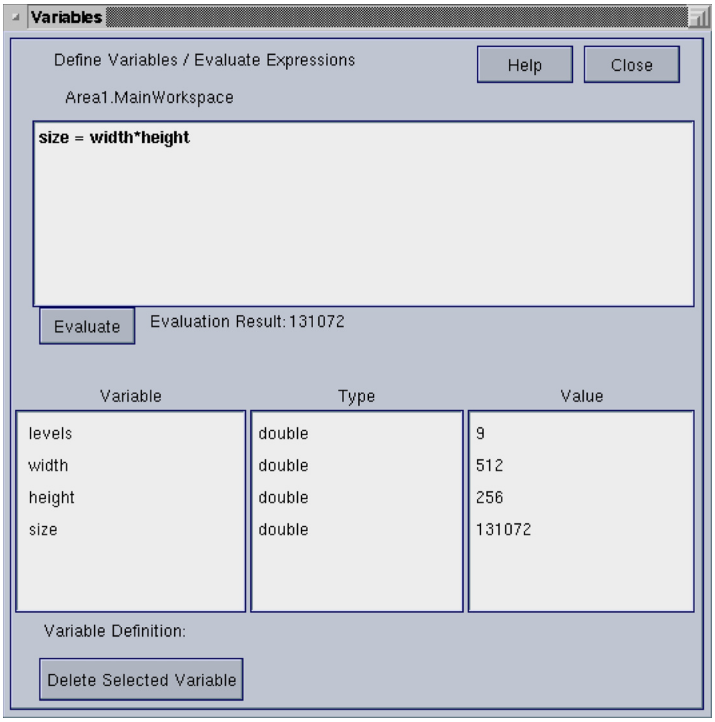
When the "Save Preferences" button is used, the enabled/disabled toolbox specification will be written to a $HOME/.kri/VQ35/ToolboxList file. The next time you run VisiQuest, this file is read in, and toolboxes will be enabled and disabled accordingly.

## Variables and Expressions

The "Variables" subform under the Workspace menu is used to define variables and evaluate expressions within the VisiQuest Visual Programming Environment programming environment. Once defined, variables and expressions can be used in place of string, integer, float, and double arguments of glyphs. The use of variables and expressions is required by many control structures; the Variables window can also be used to query the values of variables being used in control structures, and to change the values of those variables.

VisiQuest Visual Programming Environment includes a simple, textual programming language for defining variables, mathematical expressions, and functions. A variable in VisiQuest Visual Programming Environment is much the same as a variable in any other programming language; it has a value and can be referred to in expressions. An expression is a sequence of variables, literal values, function calls, and arithmetic operators. VisiQuest Visual Programming Environment contains a large number of built-in functions that the user can call. In addition, user-defined functions can be created and called.

Variables have three attributes: a definition, a data type, and a current value. The definition is simply the last expression that was assigned to it in the Variables window. The current value may be different from the value specified in the definition if the variable has been changed by Expression or Loop glyphs in the workspace. When a workspace is saved, the definition each variable in the workspace is saved. When a workspace is reset or re-loaded, the definitions of its variables are evaluated to initialize the the workspace variables.



```
size = width*height
```

Evaluation Result: 131072

| Variable | Type | Value |
|---|---|---|
| levels | double | 9 |
| width | double | 512 |
| height | double | 256 |
| size | double | 131072 |

Variable Definition:

To set variables or to enter an expression, place the cursor in the text box at the top of the Variable window, and type the expression. Press the "Evaluate" button to evaluate the expression. The value returned by the mathematical expression parserwill be displayed as the "Evaluation Result".

The three lists below the Evaluation Result, from left to right, give the variable name, the variable data type, and the current value of the variable.

Valid expressions may include variables, standard arithmetic and logical operators, literal constants (like "21"), predefined constants (e.g. "pi"), and function references. Any string of alphanumeric characters, beginning with a letter, may be used as a variable.

## Defining Variables

Unlike programming languages like C and Pascal, variables are not explicitly declared. Instead, a new variable is created by assigning a value to it. To define a variable, enter an assignment statement of the following form:

variable = expression

When this assignment statement is evaluated, the expression on the right side of the equal sign is evaluated and the result is stored in the variable. Whenever the variable appears in an expression its value is retrieved and used in place of the variable reference. Using a Pascal-style assignment statement, "x := y", selects an alternate method of defining a variable that will not evaluate the expression on the right side, but will instead save the expression with the variable. In this case, whenever the variable is used in an expression, the expression is re-evaluated and the resulting value used in place of the variable reference. If the expression contains references to other variables, the current value of the variable will depend on the current value of the referenced variables.

The example below illustrates these concepts. The variable "foo" is assigned a value of 100 using a standard (immediate) assignment statement. "bar" is assigned a value that depends on foo, also using an immediate assignment. Finally, "baz" is assigned a value that depends on foo using a deferred assignment statement.

```
foo = 100
bar = foo + 10   (bar = 110)
baz := foo + 10  (baz = 110)
foo = 200        (foo = 200, bar = 110, baz = 210)
```

To check the value of a defined variable or to evaluate an expression, enter the variable or expression in the "Expression" selection and click "Evaluate". The value will be displayed as the "Evaluation Result". Note that you must delete the previous entry before typing in a new entry. White space is ignored.

For example, to set x equal to 10, type "x = 10" in the "Expression" text parameter box, then click on "Evaluate" to enter it. The result, "10," will be displayed as the "Evaluation Result" as well as in the variable Value list.

## Data Types

Variables can contain either numeric values or character strings. Variables are not declared to be of any particular type and can be assigned either kind of value. String constants are enclosed in double quotes. When evaluating expressions, conversion between data types is performed as necessary. The example below illustrates mixing data types in expressions. The "+" operator performs simple addition, while the "." operator concatenates strings (note that the "." operator must have a space on either side of it).

```
x = 10 (x has a numeric value of 10)
str = "20" (str has a string value of "20")
x + str (yields 30)
x . str (yields "1020")
```

## Incrementing/Decrementing Variables

Use of automatic increment/decrement expressions can be used to modify the value of a variable without changing its original definition.

- To increment x by 1, enter "x++" and click on "Evaluate".
- To increment x by n (where n is a number), enter "x += n" and click on

"Evaluate".

- To decrement x by 1, enter "x--" and click on "Evaluate".
- To decrement x by n (where n is a number), enter "x -= n" and click on "Evaluate".

## Checking Values of Variables

To check the value of x, enter "x" (without the quotes) in the "Expression" text parameter box and click on "Evaluate". The current value of x will be displayed as the "Result of Evaluation" as well as in the Variables Definition list.

## Evaluating Expressions Using Defined Variables

Once a variable has been defined, expressions using that variable may be evaluated. Comparisons can be done and new variables can be defined that depend on the previously defined variable. There are also a number of predefined functions that can be used on the variable.

### Example: Comparisons

To see if x is greater than 0, enter "x > 0", and click on "Evaluate". The "Result of Evaluation" is "1" for TRUE, "0" for FALSE.

### Example: Use of Predefined Functions

To determine the natural log of x, enter "ln(x)" in the "Expression" text box and click on "Evaluate".

### Example: Defining New Variables Depending on Previous Variables

To define a variable "y" that depends on x, enter "y := x + 10" in the "Expression" text parameter box.

### Example: Changing Values of Variables

To change the value of a variable, simply enter a new expression that defines the new value. For example, to set x to 100, enter "x = 100". Any variables that depend on x will have their values automatically recalculated and updated to reflect the new value of x.

## User-defined Functions

VisiQuest Visual Programming Environment allows users to define their own functions. The syntax is similar to defining variables: a function symbol and argument list followed by an equal sign and an expression that uses the variables in the argument list. For example:

```
f(x) := ln(x / 2)
g(x) := f(x) * 100
```

## Scope Of Variables

Variables are defined with respect to the workspace selected by name from the Variable list. In workspaces having no hierarchy (ie, no procedures, no while loops, and no count loops), all variables are defined with respect to the main VisiQuest Visual Programming Environment workspace. These are considered global, and can be used by all procedures and loops within the main workspace. Variables defined within a particular procedure or loop may be used only within that procedure or loop, as well as in any subordinate procedures or loops.

Each subordinate workspace may have its own local variables. Subordinate workspaces are listed by name in the variables list, along with global variables. When a "+" symbol appears in front of the name, the variables in that workspace will not be shown. Double clicking on the "+" symbol will "open" the subordinate workspace's variable list; the "-" symbol will appear in front of the name, and the variables that are local to that workspace will appear, indented, below the workspace name. Double clicking on the "-" symbol will "close" the workspace's variable list. In this way, you may access variables that are local to a subordinate workspace.

For example, suppose the main workspace contains defined variables x, y, and z. The main workspace contains a count loop, which uses the variable i to iterate, and a while loop, which uses the variable j to iterate. The count loop contains a procedure, which has defined variables a and b. This scenario implies three levels of scope:

1. The main VisiQuest Visual Programming Environment workspace "MainWorkspace" (variables x, y, and z),
2. **a.** The count loop, named by default "Count Loop" (variable i),
   **b.** The while loop, named by default "While Loop" (variable j),
3. The procedure, named by default "Procedure" (variables a and b).

Being at the lowest level, the procedure can use all variables from the workspaces "above" it. It gets a and b from its own local definitions; it can also access the value of i from its count loop parent, and it can use x, y, and z from the main workspace. The only variable to which it does not have access is "j" from the while loop that is not part of its heritage.

At the middle level, the count loop can utilize its own variable i, as well as the x, y, and z of the main workspace. It does not, however, have access to the definition of j (defined by the while loop), or the a and b variables used by the procedure. Similarly, the while loop can utilize its own variable j, as well as the x, y, and z of the main workspace.

At the top level, the main workspace has access only to x, y, and z.

If an attempt is made to evaluate an undefined variable, an error message reading "Error! Unknown variable or constant 'variable'"will be displayed.

## Variables used by Loops and Expression

## Glyphs

A variable does not have to be defined using the Variables window before it can be used in a loop or an expression glyph.  Variables without previous definitions can be defined directly using the pane of a loop or an expression glyph; such variables will not be defined (and will not appear on in the lists of the Variables window) until the workspace is run for the first time.

For example, suppose a visual network contains an Expression glyph that sets "n = 0".  Later, the value of "n" is incremented using another expression glyph inside a loop.  It is not necessary to use the Variables window to define n, as it will be defined at the scope of the workspace in which the expression glyph appears as soon as the network is executed and the expression glyph is encountered for the first time.  The value of n, however, can be monitored with the Variables window as it is updated by the expression glyph inside the loop.

## Using Variables with Filenames or Strings

Variables can be used to specify filenames or string parameters to glyphs. One common use of this is when a VisiQuest Visual Programming Environment network is looping through a series of input or output files with numbered prefixes or postfixes.  When used as part of filenames or strings, variables must be preceded with a dollar sign ($).  For example, suppose a visual program is to be run on a set of 15 input files named "data.1", "data.2", "data.3", ...  "data.15".  The files are referenced by a glyph contained in in a count loop that uses the variable "i".  In the glyph's user interface pane, specify "data.$i" (without the quotes) as the filename.

Each time the loop executes the glyph, VisiQuest Visual Programming Environment replaces the variable reference in the filename with the current value of the variable.  This generates the sequence of filenames "data.1", "data.2", etc.

One problem with this example is that when the files are listed, they will not be in numeric order, i.e. "file.11" will appear before "file.2". This can be avoided by using variable formatting syntax when referring to variables within filenames or strings.  This syntax is: a dollar sign, a left squiggly bracket, the variable name, a colon, a printf-style format specifier (without the percent sign), and a right squiggly bracket.  For example, if the value of i is 8:

```
file.$i -> file.8
file.${i:04d} -> file.0008
file.$(i:4d}  -> "file.   4"
```

Two final notes on variables within strings and filenames:

- First, if a variable reference in a string or filename cannot be resolved, VisiQuest Visual Programming Environment attempts to look it up in the user's environment, e.g. "$HOME".  If that also fails, the reference is left unchanged inside the string.
- Second, if you need to include a dollar sign in a string, place a back-slash in front of it.  Here are some more examples.

```
$HOME/file.$i  ->  /users/frank/file.10
file.$undefined  ->  file.$undefined
"Price is \\$$price" -> "Price is $12.34"
```

## Deleting Variables

To delete a variable, select the variable to be deleted from the Variable list and then click "Delete Selected Variable".

Note that any variables depending on the deleted variables will have to be redefined; the value of a variable that depends on an undefined variable is itself undefined.

## Predefined Constants

Predefined constants recognized by the VisiQuest mathematical expression parser are summarized below.

| Constant | Value | Description |
|---|---|---|
| pi | 3.14159265358979323846 | pi |
| pi_2 | 1.57079632679489661923 | pi/2 |
| pi_4 | 0.78539816339744830962 | pi/4 |
| sqrtpi | 1.77245385090551602730 | square root of pi |
| sqrt2 | 1.41421356237309504880 | square root of 2 |
| sqrt1_2 | 0.70710678118654752440 | square root of 0.5 |
| e | 2.71828182845904523536 | natural number e |
| log10e | 0.43429448190325182765 | common log of e |
| ln2 | 0.69314718055994530942 | natural log of 2 |
| ln10 | 2.30258509299404568402 | natural log of 10 |
| log2e | 1.4426950408889634074 | log base 2 of e (1/ln2) |
| deg | 57.29577951308232087680 | radians to degrees (180/pi) |
| gamma | 0.57721566490153286060 | Euler's constant |
| phi | 1.61803398874989484820 | |
| maxfl | 1.0e+38 | a large floating point value |
| maxint | 2147483647 | largest 32 bit integer value |
| maxshort | 32767 | largest 16 bit short value |

## Predefined Functions

The standard functions recognized by the VisiQuest mathematical expression parser are summarized below.

| Function | Definition | Description |
|---|---|---|
| sin | sin(x) | sine of x |
| cos | cos(x) | cosine of x |
| tan | tan(x) | tangent of x |
| sinh | sinh(x) | hyperbolic sine of x |
| cosh | cosh(x) | hyperbolic cosine of x |
| tanh | tanh(x) | hyperbolic tangent of x |
| asin | asin(x) | arc sine of x (range -J/2 to J/2) |
| acos | acos(x) | arc cosine of x (range 0 to J) |
| atan | atan(x) | arc tangent of x (range -J/2 to J/2) |
| atan2 | atan(y,x) | arc tangent of y/x (range -pi to pi) |
| sqrt | sqrt(x) | square root of x |
| hypot | hypot(x,y) | Euclidean distance (returns sqrt(x*x + y*y)) |
| exp | exp(x) | exponential function of x |
| expm1 | expm1(x) | (exp(x) - 1) even for small x |
| ln | ln(x) | natural log of x |
| log10 | log10(x) | log base 10 of x |
| log1p | log1p(x) | log(1+x) even for small x |
| gamma | gamma(x) | log gamma function of x |
| abs | abs(x) | absolute value of \|x\| |
| floor | floor(x) | largest integer not greater than x |
| ceil | ceil(x) | smallest integer not less than x |
| rint | rint(x) | |
| pow | pow(x,y) | x raised to the power of y |
| fact | fact(x) | integer factorial of x |
| gnoise | gnoise(x) | gaussian noise function |
| unoise | unoise(x) | uniform noise function |
| impulse | impulse(x) | impulse function |

| Function | Definition | Description |
|---|---|---|
| step | step(x) | unit step function |
| sign | sign(x) | sign function |
| j0 | j0(x) | bessel functions |
| j1 | j1(x) | bessel functions |
| y0 | y0(x) | bessel functions |
| y1 | y1(x) | bessel functions |
| erfc | erfc(x) | error function for 1.0 - erf(x) |
| erf | erf(x) *where erf(x) = 2/sqrt(pi)*integral from 0 to x of exp(-t*t) dt.* | error function of x |

The following functions take a string argument and return a new string.

| Function | Definition | Description |
|---|---|---|
| make_upper | make_upper(str) | make upper case |
| make_lower | make_lower(str) | make lower case |
| dirname | dirname(path) | get the directory part of a path |
| basename | basename(path) | get the filename part of a path |
| extension | extension(path) | get the file part of a path |

## If/Else Expressions

There are two kinds of if/else expressions, a condition expression and an if-expression. They both have the same effect but use a different syntax. The conditional expression consists of an expression followed by a question mark and two expressions separated by a colon.

```
expr1 ? expr2 : expr3
```

If expr1 evaluates to a non-zero value, expr2 is evaluated, otherwise expr3 is evaluated. The if-expression accomplishes the same thing with a slightly less obscure syntax.

```
if (expr1) expr2 else expr3
```

The following examples both set "i" and performs a conditional increment of the variable i, (i = (i+1)%11).

```
i = (i > 10) ? 0 : i + 1
i = if (i > 10) 0 else i + 1
```

### User Defined Functions

You may also define your own function, using a method similar to that of defining variables. In some instances, this proves to be easier than trying to keep track of many dependent expressions or evaluating complex expressions.

For instance:

```
f(x,y)= cos(sqrt(x**2+y**2))*exp(0.0-sqrt(x**2+y**2)/4)
```

To evaluate the above expression at (x = 0) and (y = 0), simply enter "f(0,0)" which should result in the value of "1.0". You may use predefined or previously-defined expressions as well as variables in evaluating the function "f". For example, it is valid to enter "f(-pi*2,2+x)" for evaluation. In this example evaluation, the "x" in "f(x,y)" is explicitly defined as "-pi*2" and the "y" in "f(x,y)" is explicitly defined as "2+x." Contrastingly, if you enter an expression that is dependent on variables not explicitly defined within the function, as in "f(x,y)," then the variable list will be searched for the current values of the variables used.

```
f(x) = x + z
```

The variable z is not defined by the function "f" so the current variable list will be searched (if the variable is not defined then an error will be displayed).

Recursive definitions are allowed; however, it is important to understand that there are no checks on the stack size. Thus, it is possible to core dump the application using recursive definitions. With any recursive function, you must to make sure that the recursive expression has a base case in order to prevent a stack overflow from occurring.

For example, the intuitive way to define a Fibonacci sequence is with the

```
fib(n) = (n == 0 || n == 1) ? n : fib(n-1) + fib(n-2)
```

The Fibonacci sequence of 10, "fib(10)", should return 55. The base case occurs when n equals 0 or 1. However, if you enter "fib(-2)", a stack overflow will occur, core dumping the application. Consider, instead, the following definition of the Fibonacci sequence:

```
fib(n) = (n > 1) ? fib(n-1) + fib(n-2) : n
```

Although this is not the strict definition of a Fibonacci sequence, this function will protect the stack from overflowing when a negative number is used as the argument.

## *Logical Operators*

The logical operators used by the VisiQuest mathematical expression parser to evaluate logical expressions are summarized below. The result of an evaluation will be returned as "1" for TRUE and "0" for FALSE.

| Logical (C style) | Logical (Fortran style) | Description |
|---|---|---|
| == | .EQ | See if two expressions are equal |
| > | .GT | See if first expr is greater than second |
| < | .LT | See if first expr is less than second |
| >= | .GE | See if first expr is greater/equal than second |

| Logical (C style) | Logical (Fortran style) | Description |
|---|---|---|
| <= | .LE | See if first expr is less/equal than second |
| != | .NE | See if first expr is not equal to second |
| && | .AND.:AND | AND two expressions together |
| \|\| | .OR.:OR | OR two expressions together |
| ! | .NOT.:NOT | NOT one expression |

## *Arithmetic Operators*

Valid expressions recognized by the VisiQuest mathematical expression parser may include the standard arithmetic operators summarized below.

| Symbol | Meaning | Description |
|---|---|---|
| * | Multiply | multiple the two expressions together |
| / | Divide | divide the first expression by the second |
| - | Subtract | subtract the second expression from the first |
| + | Add | add the two expressions together |
| ** | Power | take the first expression to the power of the second |

### Arithmetic Bit-wise Operators

Valid bit-wise expressions may include the following standard C style operators. Note: all expressions will be evaluated and then cast to integers in order to do the arithmetic bit-wise operation.

| Symbol | Meaning | Description |
|---|---|---|
| << | Shift Left | shift the first expr to the left by second |
| >> | Shift Right | shift the first expr to the right by second |
| & | Bit wise AND | and the two expressions together |
| \| | Bit wise OR | or the two expressions together |
| ^ | Bit wise XOR | exclusive or two expressions together |
| ~ | Bit wise INVERT | invert the expression |
| % | Bit wise MODULOS | modulos the expression |

## Arithmetic Assignment Operators

Arithmetic assignment operators are used to update a current expression to a new expression. The new and old expressions are operated on by the type of assignment operator. An example is the "+=" operator which adds the new expression to the existing expression. Valid assignment expressions may include the following standard C style operators.

| Symbol | Meaning | Description |
|---|---|---|
| *= | Multiply | multiply the current expr by the original expr |
| /= | Divide | divide the current expr from the original expr |
| -= | Subtract | subtract the current expr from the original expr |
| += | Add | add the current expr to the original expr |
| <<= | Shift Left | shift the first expression to the left by second |
| >>= | Shift Right | shift the first expression to the right by second |
| &= | Bit wise AND | and the current expr with the original expr |
| \|= | Bit wise OR | or the current expr with the original expr |
| ^= | Bit wise XOR | exclusive or the current expr with the original expr |

## String Operators

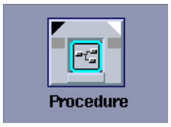
VisiQuest Visual Programming Environment supports the following string operators and assignment operators.

| Symbol | Meaning | Description |
|---|---|---|
| \. | Concatenate | concatenate two strings |
| \.= | Concatenating assignment | concatenate onto a variable |

Examples

```
str = "Hello" . " " . "world."
str .= "  Just testing."
```

# Procedures



Procedure glyphs differ from "regular" glyphs in that they have a "procedure" pixmap in the middle and an "Open Workspace" button at the top-right corner.

Hierarchy is supported within the VisiQuest Visual Programming Environment in the form of procedures. Similar in concept to a subroutine in a textual programming language, a visual programming procedure allows you to modularize a visual program so that certain operations are confined to a particular location within the visual program. Procedures promote readability of a visual program. In addition, they allow a certain portion of a network that performs a particular function to be used multiple times within a visual program. The use of procedures is especially effective with large and complex workspaces, as well as with workspaces that are required to perform a particular task a number of times.

A procedure in this context is for use in the current workspace only; thus, it can be considered a "local" procedure. If you want a procedure that is to be shared among different VisiQuest workspaces, you should first follow the instructions given here for creation of a procedure. Then, with the procedure open, create a compiled workspace from the procedure. Once a compiled workspace has been created from the procedure, it can be considered "global"; that is, the procedure will be accessible from the Glyph Menus, Accelerated Routines List, and Accelerated Finder list just like any other operator, and can be used by a number of different visual programs.

## *Creating Procedures*

If there are no inputs connected to the set of glyphs when the procedure is created, then the procedure will have no inputs ports. So the user should make sure that there is at least one glyph with an incoming connection that extends beyond the selection. Even if a procedure is created with no inputs, this can be fixed by editing the procedure. The user needs to click on the file folder portion of the procedure glyph. This will open the procedure in the VPE for editing. Then open the pane for the glyph whose inputs they want to export. In the pane, move the mouse over the input to be exported. It will turn yellow, and a pop-up message will appear instructing them to right-click the mouse in order to export this parameter. At that point they can click the right mouse button, and select Export. This will cause the parameter to be exported, and an input data path can now be connected to the procedure.

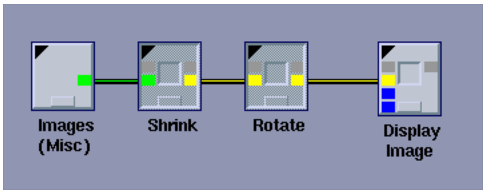
To create a procedure:

1. First, use rubberbanding to select the set of glyphs on the workspace that are to be incorporated into the procedure. Be sure that only those glyphs that are to be made part of the procedure are selected.

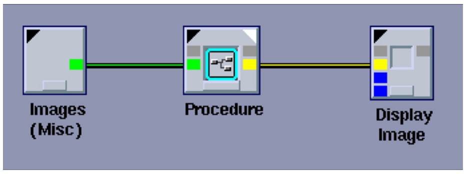
2. Select "Create Procedure" from the Control menu (or use the appropriate icon from the workspace command bar to create the procedure). The selected glyphs will be removed from the workspace in preparation for procedure

creation. Move the cursor into the workspace; it will be changed to the "busy" cursor while the procedure is being created. When the procedure glyph is ready to be placed, the cursor will be changed to the "arrow" cursor.

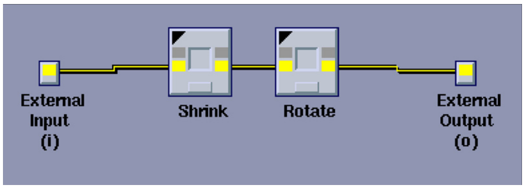
3. Place the new procedure by clicking at the desired location in the workspace. After you place the procedure glyph, the network connections will be automatically redrawn with respect to the new procedure.



In a very simple network, two glyphs have been selected to be incorporated into a procedure.



The procedure has been created to contain the "Shrink" and "Rotate" glyphs.



Inside the open procedure containing the "Shrink" and "Rotate" glyphs.

## Opening/Closing Procedures

To open the procedure, click on the "Open Workspace" button that appears in the upper-right corner of the glyph. The main VisiQuest Visual Programming Environment workspace, displaying the "outer" visual program, will be replaced with the procedure workspace, displaying the network that comprises the procedure. Note that the workspace name label that appears below the workspace will change from "Active Workspace: MainWorkspace" to "MainWorkspace.Procedure"; this label helps to remind you of your "location" within the visual program at any given time.

When the procedure workspace is open, all operations initiated through use of the workspace Command Bar, the "Workspace" menu, and the "Edit" menu will apply to the procedure rather than to the main VisiQuest Visual Programming Environment workspace.

To close the procedure workspace, click on the "Close" minus-sign that appears in the upper-left corner of the procedure workspace.  The procedure workspace will be replaced with the main VisiQuest Visual Programming Environment workspace once again.

You can also click the up-arrow that appears in the upper-right corner of the procedure workspace to create a top-level window for the procedure and to move the workspace to a another location on the screen; moving the top-level workspace allows the procedure glyph in the main VisiQuest Visual Programming Environment workspace beneath to be seen.  Click the now-inverted white triangle in the upper-right corner of the procedure glyph to close the procedure workspace.

### *Renaming Procedures*

By default, the name of a newly created procedure is "Procedure."  We recommend improving the readability of the visual program by giving the procedure a more descriptive name. To change the name of a procedure, click on the current name, "Procedure," that appears underneath the procedure glyph; a pop-up prompt in which you can enter a new name displays.  Enter the desired name in the string selection that appears in the popup prompt, and click "OK." Note that this process can be used to rename any glyph, not only procedure glyphs.

### *Copying Procedures*

Procedures may be copied or duplicated using the editing features of VisiQuest Visual Programming Environment.  However, it is important to be aware that once a procedure is copied or duplicated, the copied glyph is separate and independent from the original.  For example, if you make multiple copies of a procedure, and then make a modification to one of the copies, the modification will not be reflected in any of the other procedures.

### *Nesting Procedures*

If desired, procedures may be nested.  Creation of nested procedures may be done in two ways: (1) If one or more procedures has already been incorporated into a visual program, repeating the procedure creation process in the main VisiQuest Visual Programming Environment workspace and including an existing procedure glyph in the new procedure will cause the existing procedure to be nested inside the newly created procedure; (2) If an existing procedure is open, repeating the procedure creation process inside the procedure workspace will cause the newly created procedure to be nested inside the existing procedure.

## Control Structures

Control structures are available from the Create menu.  They allow you to create more complex visual programs by providing constructs that you may use to direct the flow of your visual program. In this context, "flow" refers to both "control flow" and "data flow;" when there is more than one possibility for data flow, control structures are used to determine which path the data flow will take.

Control structures fall into two major categories: conditional constructs and looping constructs.  Both types of constructs are discussed in detail below.

Conditional constructs imply that control flow of the visual program will be directed one way or another, depending on the value of variables defined for the workspace.  The available conditional constructs include:

- If/Else

  Directs data flow to one path if a specified condition is met, to another path if the condition is not met.
- Merge

  Directs data flow from two separate paths to the same path, whether data arrives from the first path, the second path, or both; no condition is involved.
- Switch

  Selects one of two inputs for use by the visual network, depending on the value of a conditional statement.
- Trigger

  Delays execution of a glyph until data is made available by another glyph which is not otherwise connected to the dependent glyph.
- Expression

  Sets the value of a variable; used in conjunction with other control structures for purposes of variable initialization and/or modification.

In contrast, looping constructs cause control flow of the visual program to iterate through a particular section of the network multiple times.  Looping constructs depend on the value of variables to determine how many times the control flow will iterate on the specified section of the network.  The available looping constructs are:

- Count Loop

  Causes the data flow to iterate through a particular section of network a specified number of times.
- While Loop

  Causes the data flow to iterate through a particular section of network as long as a particular condition is met.

## *Creation of a Control Glyph*

Control structures in VisiQuest Visual Programming Environment are represented by special purpose glyphs.  The glyphs representing control structures are accessible from the Control menu; selecting one of the items in this menu will cause the corresponding glyph to appear in the workspace.

## Conditional Constructs

### If/Else

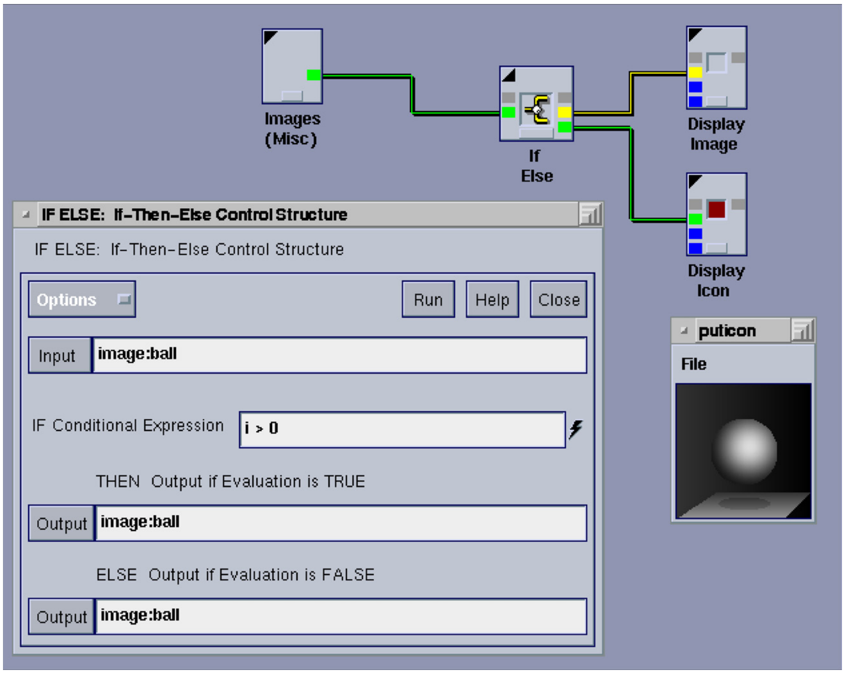
The if/else conditional construct is used to divert data flow in a visual network into one of two paths.  Data flow will be directed to the upper path if a specified condition is TRUE; it will follow the lower path if the specified condition is FALSE.

When data is available at the input connection of the if/else glyph, the specified expression is evaluated.  If the expression is evaluated as TRUE, the input connection data is passed to the upper (TRUE) output connection; if the expression is evaluated as FALSE, the input connection data is passed to the lower (FALSE) output connection. Data flow will then progress downstream from the if/else glyph to the rest of the network.

To use the if/else conditional construct:

1. Begin by creating an if/else glyph.
2. Use the "Variables" window to define and initialize the variable that will be used in the conditional expression for the if/else conditional construct, if it has not already been defined.
3. Open the pane for the if/else glyph and type in the expression that defines the condition that will determine how the data flow is to be directed.  Be sure to provide an expression that will evaluate as boolean (TRUE or FALSE).
4. Close the pane, and connect the if/else glyph at the desired location in the visual program.

When the data flow of the visual program reaches the if/else glyph, the specified condition will be evaluated and the data will be propagated to one of the two output connections, depending on whether the expression was evaluated as TRUE or FALSE.



IF ELSE: If-Then-Else Control Structure

Input: image:ball

IF Conditional Expression: i > 0

THEN Output if Evaluation is TRUE

Output: image:ball

ELSE Output if Evaluation is FALSE

Output: image:ball

A visual network containing an if/else glyph. The pane for the if/else glyph is displayed to the lower left.

In the example above, the variable "i" has been defined and set to 0 using the "Variables" window. The condition specified on the pane for the if/else glyph is "i > 0". If the condition evaluates as TRUE, the input image is passed to the first output of the if/else glyph, which is the "Display Image" glyph. However, since "i" is set to 0, the condition is evaluated as FALSE; accordingly, the input image is routed to the second output of the if/else glyph, which is the "Icon" glyph. Thus, the result of the visual program is that the input image displays as an icon. Note that when the conditional expression of the if/else glyph is evaluated as FALSE, the fact that data will flow to the second output as a result of the evaluation is reflected by the pixmap on the if/else glyph, which is highlighted to reflect "connection" between the input and the second output.

If the variable "i" is modified to have a value greater than 0, and then the visual network is reset and rerun, then the data flow in this network will take the upper path, and the input image will be displayed at full size.

## Merge

The merge control construct is used to merge two paths of a visual program into one.

When data is available at either of the input connections of the merge glyph, the data flow will be directed to the single output connection.

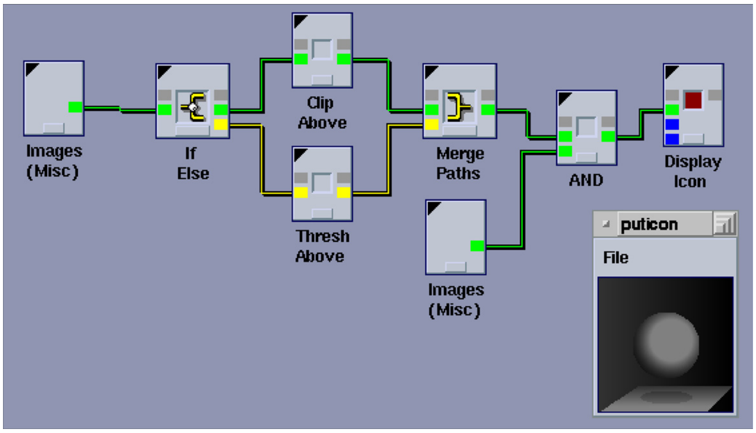
To use the merge control construct:

1. Begin by creating a merge glyph.
2. Connect the output of the last glyph in the first path to the first input of the merge glyph; connect the output of the last glyph in the second path to the second input of the merge glyph.

The operation of the merge glyph will differ depending on whether or not it is possible for data to flow across both paths at the same time.

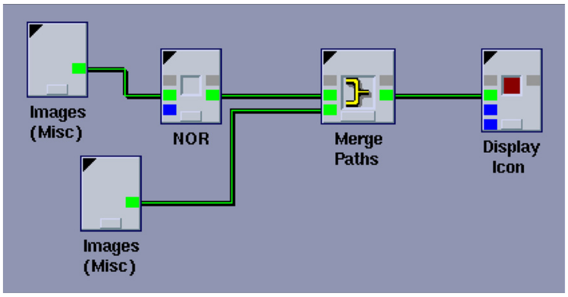
In the first case, when the two independent paths have been created with the use of an if/else glyph, it is not possible for data to flow across both paths at once; either one path will be active, or the other one will, but not both. In this case, the merge glyph is simply used to rejoin the two paths into one; this approach is easier and visually more efficient than the alternative, which is to replicate the downstream portion of the visual network and use identical segments for each path. When data becomes available to the merge glyph, regardless of which path was determined by the if/else glyph, the input is simply propagated directly to the output of the merge glyph and the data flow continues downstream in a single path.

In the second case, when the two independent paths have separate origins and are not controlled by some conditional construct, it is possible for both data flow paths to be used simultaneously. When this is the case, the merge glyph will propagate the data from either path as soon as it becomes available. In the case that both paths produce available data at the same time, the first input will be passed to the output; immediately following it, the second input will be passed to the output.  Thus, the data flow of the two paths will "take turns" moving through the merge glyph.



A visual network containing an merge glyph used to reconnect two data flow paths produced by an if/else glyph.

In the example illustrated above, a merge glyph is used to reconnect the two data flow paths produced by an if/else glyph. If the variable "i" evaluates to an even number (ie, if "(i%2) == 0" evaluates as TRUE), the "Floating Ball" input image will be directed into the "Clip Above" glyph.  If the variable "i" evaluates to an odd number (ie, if "(i%2) == 0" is evaluated as FALSE), the input image will be directed into the "Thresh Above" glyph.  The merge glyph is used to direct the resulting output into the "AND" glyph, regardless of whether the input image is clipped or thresholded.  The clipped (or thresholded) image is then AND-ed with the original floating ball, and the result is displayed in iconic form.  In the example illustrated above, the variable "i" had been set to 10, so that the input image was clipped.



A visual network containing an merge glyph used to connect two data flow paths with different origins.

In the example illustrated above, a merge glyph directs two separate data flow paths into the "Display Icon" glyph, one after the other. The top path has the "Floating Ball" input image being processed by a "NOR" operator; the bottom path simply provides the "Spanish Gull" input image with no data processing.  Since the bottom path will execute faster, the icon will first display the gull image.  After a slight delay while the ball input image is processed by the "NOR" operator, the icon will be updated to display the modified ball image.

## Switch

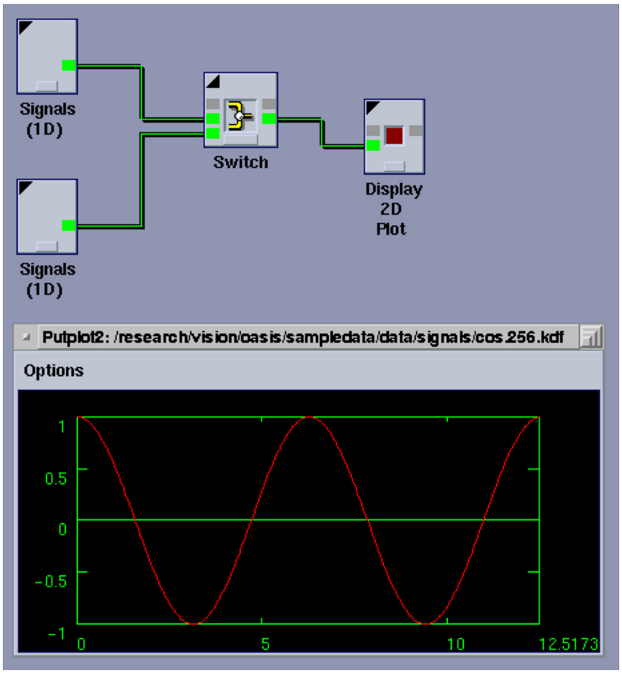
The switch control construct is used to specify which of two inputs will be used by a visual network, depending on a conditional expression.  Data will be obtained from the first input if a specified condition is TRUE; it will be obtained from the second path if the specified condition is FALSE.

When data is available at both of the two input connections of the switch glyph, the specified expression is evaluated.  If the expression is evaluated as TRUE, the data is passed from the first (TRUE) input connection of the switch to the output of the switch.  If the expression is evaluated as FALSE, the data from the second (FALSE) input connection of the switch to the output of the switch.

To use the switch control construct:

1. Begin by creating an switch glyph.
2. Use the "Variables" window to define and initialize the variable that will be used in the conditional expression for the switch control construct if it has not already been defined.
3. Next, open the pane for the switch glyph and type in the expression that defines the condition to determine how the data flow is directed. Be sure to provide an expression that will evaluate as boolean (TRUE or FALSE).
4. Close the pane of the switch glyph.
5. Connect the switch glyph at the desired location in the visual program.

When the data flow of the visual program reaches the switch glyph, the specified condition will be evaluated and the data will be passed from one of the two input connections to the output connection, depending on whether the expression was evaluated as TRUE or FALSE.

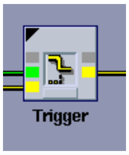


A visual network containing a switch glyph.

The condition specified on the pane for the switch glyph is "(i % 2) == 0". If the condition was evaluated as TRUE, the first input to the switch, signal "plot2d:brain," would be passed to the output of the switch glyph and would display as a 2D plot. However, since the current value of "i" is set to 5, the condition is evaluated as FALSE, and, the second input to the switch, signal "plot2d:cosine," is passed to the output of the switch glyph. Therefore, the result of the visual program is that the cosine curve is displayed as the 2D plot.

If the variable "i" is modified to have an even value rather than an odd value, and the visual network is reset and rerun, then the data flow in this network will originate from the first input to the switch, and the displayed 2D plot will be the signal of the healthy NMR brain scan.

## Trigger



The trigger control construct is used to control the scheduling of a data path. The use of a trigger control construct is similar to the use of a control connection in that it allows you to make the execution of a particular glyph dependent on the execution of a second glyph that does not supply the data flow for the first glyph.

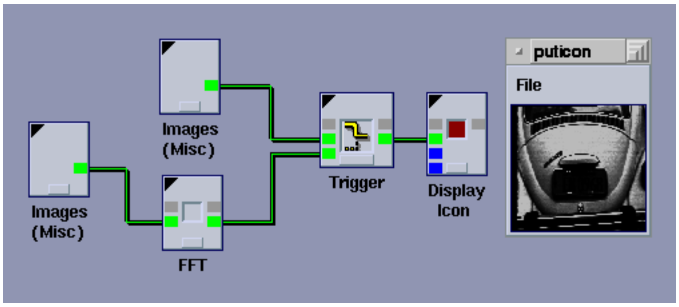
With a control connection, however, the dependent glyph cannot execute until the glyph on which it depends has completely finished executing. In contrast, a trigger is used when the dependent glyph needs to execute as soon as the glyph on which it depends makes data available, regardless of whether or not it has finished executing.

As soon as data is available at the input connection of the trigger glyph, the trigger will allow the dependent glyph to be scheduled. The dependent glyph will not be scheduled prior to that, regardless of whether its input connection has data available or not.

To use the trigger control construct:

1. Begin by creating a trigger glyph.
2. Connect the output of the glyph which passes data to the first (upper) input of trigger. Connect the output of the triggering glyph to the second (lower) input of trigger.

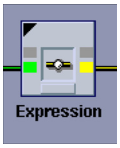
When the data flow of of the visual program reaches the trigger, its execution will be delayed until the glyph providing the input to the trigger makes data available.



A visual network containing a trigger glyph.

In the example illustrated above, the trigger glyph is used to make the display of the VW icon dependent on the output of the fft. If the "Images (Misc)" glyph was connected directly to the "Icon" glyph, the VW would be displayed as an icon immediately. However, by using the trigger glyph to intervene, the "Icon" operator will not be scheduled for execution until after the "FFT" operator has been run on its own input image, the moon. As soon as the FFT operator makes data available, the VW input image will be displayed by the icon operator.

## Expression



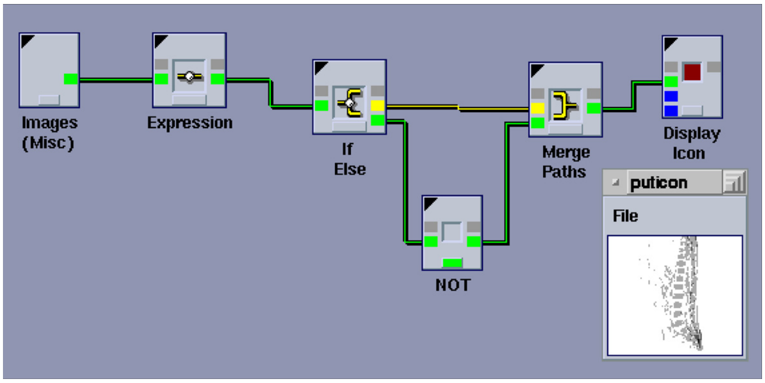
The expression control construct is used to evaluate an expression at a certain point in the network. The expression glyph provides a convenient way to reinitialize variables within a loop construct, initialize variables prior to the execution of another control construct, or to set new variables at any point in a network.

When data is available at the input connection, the specified expression is evaluated. The input connection data is then propagated to the output connection. Data flow will then progress downstream from the expression glyph to the rest of the network.

To use the expression control construct:

1. Begin by creating an expression glyph.
2. Open the pane for the expression glyph and type the expression that defines the variable, initializes the variable, or modifies the value of the variable.
3. Close the pane of the expression glyph.
4. Connect the expression glyph at the desired location in the visual program.

When the data flow of the visual program reaches the expression glyph, the specified expression will be evaluated before the data is propagated to the output connection of the expression glyph.



A visual network containing an expression glyph.

In the example illustrated above, the expression glyph is used to auto-increment the value of the variable before the conditional expression of the if/else glyph is evaluated. First, the "Variables" window was used to define the value of "i" as 0. The pane of the expression glyph is used to specify "i++" as the expression that will be evaluated. In the if/else glyph, "(i%2)==0" is the conditional expression which will evaluate as TRUE if "i" is an even number, FALSE if "i" is an odd number. If the conditional expression of the if/else glyph is evaluated as TRUE, the input "Floating Ball" image will be displayed as an icon; if it is evaluated as FALSE, the ball image will first be processed with the "NOT" bitwise arithmetic operator before it is displayed as an icon.

When executing this network for the first time, "i" is initialized to 0. Then, "i" is incremented to 1 by the expression glyph. By the time the conditional expression of the if/else glyph is evaluated, "i" is an odd number, which means that the "Floating Ball" image is sent to the "NOT" operator before being displayed as an icon.

To see the change in variable value, re-run the network with "i" initially having a value of 1, which will be incremented to 2 by the expression glyph. Since 2 is an even number, the original image of the ball will be displayed as the icon. The network can be run many times in succession; since the expression glyph increments "i" to the next integer each time the network is run, the output will alternate between the two output paths.

## Looping Constructs

Looping is supported within the VisiQuest Visual Programming Environment visual language with the use of procedures. The creation of loop constructs, opening and closing of loop constructs, and nesting of loop constructs is identical to operating with procedures. Like procedures, loop constructs in a visual program constitute hierarchy.

### Creating A Loop Construct

To create a loop construct:

1. First, select by rubberbanding the set of glyphs in the workspace that are to be incorporated into the loop construct. Be sure that only those glyphs that are to be made part of the loop construct are selected.
2. Next, create the loop construct. Depending on the type of loop construct desired, "Count Loop" or "While Loop" can be selected either from the "Create" menu or from the workspace Command Bar. The selected glyphs will be removed from the workspace in preparation for loop construct creation. Move the cursor into the workspace; it will be changed to the "busy" cursor while the loop construct is being created. When the loop glyph is ready to be placed, the cursor will be changed to the "arrow" cursor.
3. Place the new loop glyph by clicking at the desired location in the workspace. After you place the loop glyph, the network connections will be automatically redrawn with respect to the new loop glyph.

### Opening/Closing Loop Glyphs

To open the loop glyph, click on the "Open Workspace" white triangle in the upper-right corner of the glyph. The workspace displaying the loop glyph will be replaced with the loop workspace, displaying the network that comprises the contents of the loop.

To close the loop workspace, click on the "Close" minus-sign that appears at the upper-left corner of the loop workspace. The loop workspace will be replaced with the main VisiQuest Visual Programming Environment workspace once again.

You may also click the up-arrow in the upper-right corner of the loop workspace to create a top-level window for the procedure and to moveit so that the main VisiQuest Visual Programming Environment workspace beneath is visible. Clicking the now-inverted white triangle in the upper-right corner of the loop glyph will close the loop workspace. Note that the triangle returns to its original position once the workspace is closed.

### Nesting Loops

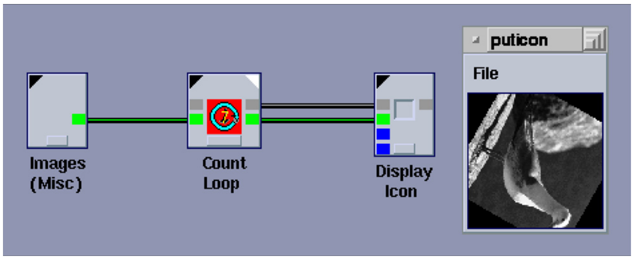
Loops may be nested. Creation of nested loops may be done in two ways: (1) If one or more loops has already been incorporated into a visual program, repeating the loop construct creation process in the main VisiQuest Visual Programming Environment workspace and including an existing loop glyph in the new loop construct will cause the existing loop to be nested inside the newly created loop construct; (2) If an existing loop workspace is open, repeating the loop construct creation process inside the loop workspace will cause the newly created loop constructs to be nested inside the existing loop.

## Count Loop

The count loop construct is used to repeat execution of a particular region of a visual network for a specified number of times.

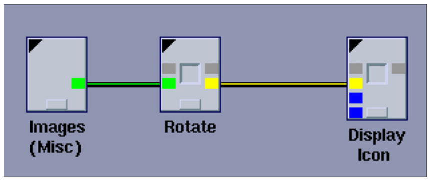
After you have created the count loop, open the pane for the count loop glyph to specify the count loop parameters (click the black triangle in the upper-left corner of the glyph to open and close the pane). Provide a variable, initial value, final value, and increment value in the appropriate selections on the count loop pane. When the count loop parameters are complete, close the pane.

When data is available at the input connection(s) of the count loop glyph, events will occur as follows. First, the count variable will be initialized. Next, the current value of the count variable will be compared to the final value specified; if the current value is less than the final value, the network contained in the count loop will be executed. After the glyphs in the count loop are executed, the increment value specified will be added to the current value of the count variable. The new value of the count variable will again be compared to the specified final value; if the current value is still less than the final value, the network will be rescheduled. This process will be repeated until the current value of the count variable meets or exceeds the final value specified. Data flow will then progress downstream from the output connection(s) of the count loop glyph to the rest of the network.

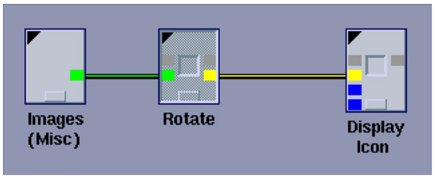


To create the example illustrated above, this is the procedure that was followed:

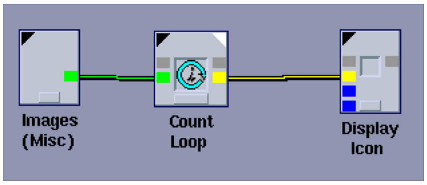
1. We first create a simple visual program consisting of an "Images (Misc)" glyph, a "Rotate" glyph, and an "Icon" glyph.  The pane of the "Images (Misc)" glyph is opened so that the input image can be set to the "Spanish Gull".



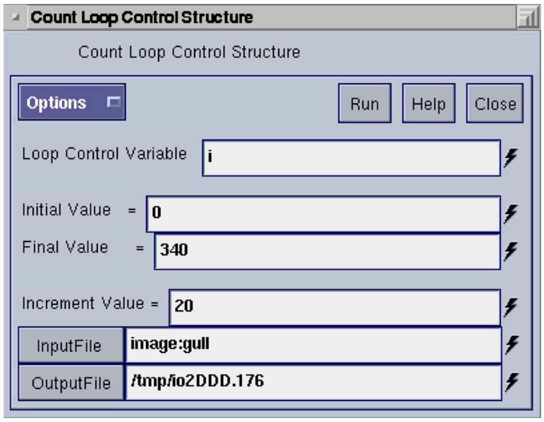
2. Next, the "Rotate" glyph is selected in preparation for creation of a count loop.



3. Click on the "Create Count Loop" command button in order to create a count loop to contain the "Rotate" glyph.  The "Rotate" glyph is replaced by the "Count Loop" glyph in the main VisiQuest Visual Programming Environment workspace.



4. The pane for the count loop is used to specify that the count loop will use the variable "i" to iterate from an initial value of 0 to a final value of 340 in increments of 20.



Count Loop Control Structure

Loop Control Variable i

Initial Value = 0

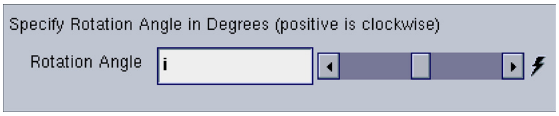
Final Value = 340

Increment Value = 20

InputFile image:gull

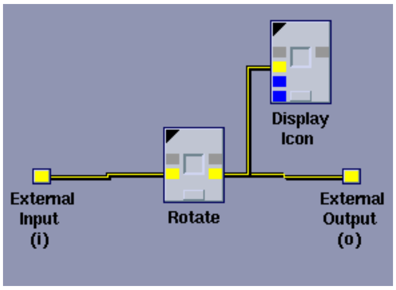
OutputFile /tmp/io2DDD.176

5. Clicking on the "Open Workspace" button of the "Count Loop" glyph displays the count loop workspace. Inside the count loop workspace, the pane of the "Rotate" glyph is used to specify that the angle used to rotate the Spanish Gull image will be determined by the value of "i."

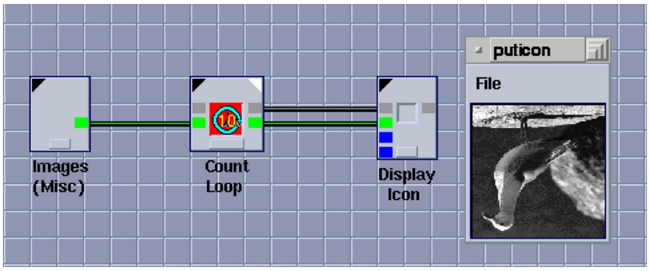

Specify Rotation Angle in Degrees (positive is clockwise)

Rotation Angle: i

6. So that we can see the intermediate results of the loop, an "Icon" glyph is created inside the count loop workspace and connected to the "Rotate" glyph.



7. Finally, the count loop workspace is closed. A control connection is used to prevent the outer "Display Icon" glyph from being executed until the loop is completed. The visual program with the count loop is executed. As the count loop iterates, the intermediate results will be displayed in icon form. The number of the count loop iteration is displayed inside the control pixmap of the count loop glyph.

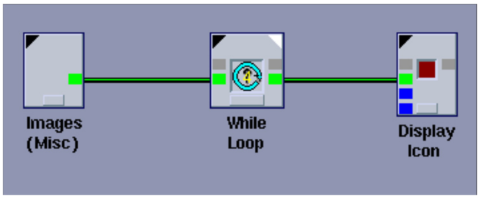


## While Loop

The while loop construct is used to repeat execution of a particular region of a visual network as long as a particular condition is met.

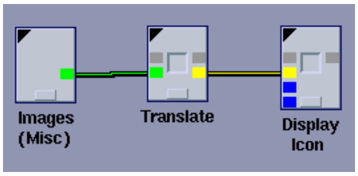
After you have created the while loop, open the pane for the while loop glyph to specify the while loop parameters (click the black triangle in the upper-left corner of the glyph to open and close the pane). Provide the conditional expression that will be used to determine how many times the while loop is to be executed. If desired, you may also provide an initial expression and an update expression in the appropriate selections on the while loop pane. When the while loop parameters are complete, close the pane.

When data is available at the input connection(s) of the while loop glyph, events will occur as follows. First, the initial expression (if provided) will be evaluated so that the variable used in the conditional expression can be initialized. Next, the conditional expression will be evaluated; if it evaluates as TRUE, the network contained in the while loop is executed. After the glyphs in the while loop are executed, the update expression (if provided) will be evaluated, so that the variable used in the conditional expression can be updated. The conditional expression will then be evaluated again; if the conditional expression is still TRUE, the network will be rescheduled; the process will be repeated until the conditional expression is evaluated as FALSE. Data flow will then progress downstream from the output connection(s) of the while loop glyph to the rest of the network.

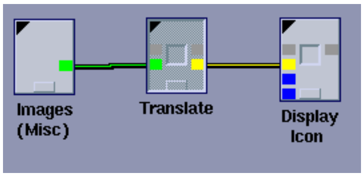


To create the example illustrated above, this is the procedure that was followed:

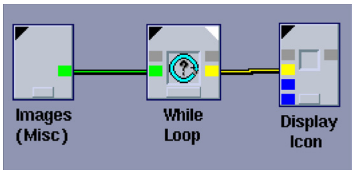
1. First, create a simple visual program consisting of an "Images (Misc)" glyph, a "Translate" glyph, and a "Display Icon" glyph. The pane of the "Images (Misc)" glyph specifies the input image as the "Moon".



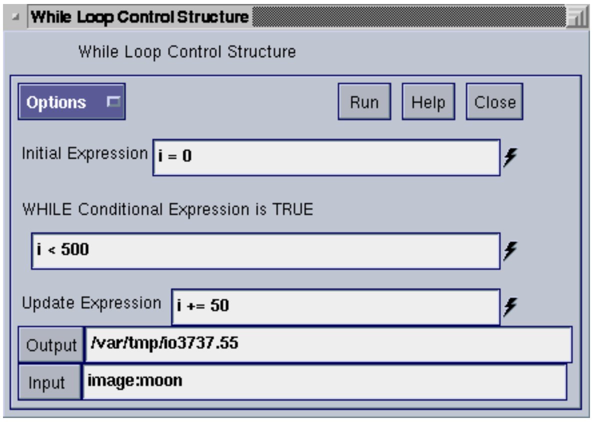
2. Next, the "Translate" glyph is selected in preparation for creation of a while loop.



3. Click the "While Loop" command button from the Create menu in order to create a while loop to contain the "Translate" glyph. The "Translate" glyph is replaced by the "While Loop" glyph in the main VisiQuest Visual Programming Environment workspace.



4. The pane for the while loop is used to specify that the while loop will use the variable "i" to iterate from an initial value of 0 to a final value of 500 in increments of 50.



While Loop Control Structure

Initial Expression: i = 0

WHILE Conditional Expression is TRUE

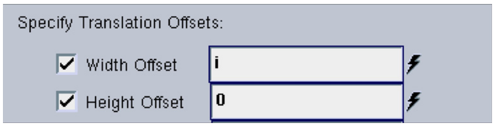
i < 500

Update Expression: i += 50

Output: /var/tmp/io3737.55

Input: image:moon

5. Clicking on the "Open Workspace" white triangle of the "While Loop" glyph displays the while loop workspace. Inside the while loop workspace, the pane of the "Translate" glyph is used to specify that the width offset used to translate the moon image will be determined by the value of "i."

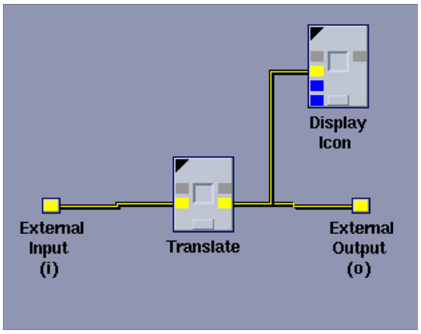


Specify Translation Offsets:

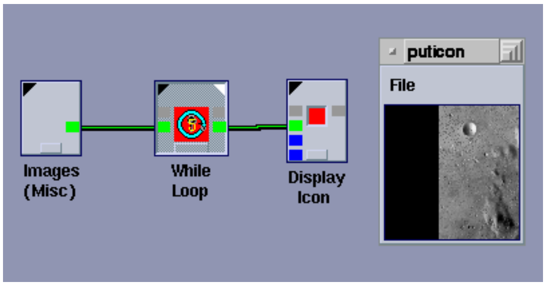
☑ Width Offset: i

☑ Height Offset: 0

6. So that we can see the intermediate results of the loop, an "Icon" glyph is created inside the while loop workspace and connected to the "Rotate" glyph.



7. Finally, the while loop workspace is closed. A control connection ensures that the outer "Display Icon" glyph will not be executed until the while loop is completed.  The visual program with the while loop is executed.  As the while loop iterates, the intermediate results will be displayed in icon form, and the number of the while loop iteration is displayed inside the control pixmap of the while loop glyph.



## Compiled Workspaces

Compiled workspace creation is a software development process, and is considered an advanced issue with respect to the use of VisiQuest Visual Programming Environment.  This section assumes that you are familiar with the use of VisiQuest as a software development system and that you have read the Toolbox Programming Manual. It is further assumed that you have experience with the creation/modification of a graphical user interface, are familiar with the use of craftsman and composer, and that you understand the process of software object creation and maintenance within the VisiQuest System.  If you do not have the necessary background in these areas, it is recommended that you read Chapters 1 - 6 of the Toolbox Programming Manual before continuing.

Once you have completed a visual program, you may create from it a stand-alone application that can be run independently of VisiQuest Visual Programming Environment. Such a program is called a compiled workspace. A compiled workspace can be brought into VisiQuest Visual Programming Environment as a new operator. It is accessible via the Glyph Menus, the Accelerated Routines List, and the Accelerated Finder list like any other VisiQuest program.

Compiled workspaces provide re-usability for visual programs. In addition, compiled workspaces are a convenient method of "packaging" visual programs for others who may not be familiar with visual programming to use the solution without being concerned, or even aware, of its origin as a visual program.

Because a compiled workspace eliminates the overhead associated with creating a VisiQuest workspace with glyphs, connections, and the like, it is also more efficient than the visual program from which it originated. Because no interaction is required from a user (unless there are glyphs in the network that require user interaction), the compiled workspace also has the capability of being run automatically. For example, the compiled workspace might be used as part of a test suite that is executed repeatedly with different input parameters by a script that is run overnight.

Note that while a compiled workspace is an independent program it will require the binary for each glyph used in its network in order to run. In order to package a compiled workspace for execution in an environment where VisiQuest is not installed, a packaging tool will be required. As of August 2, 1999, this tool was not yet available. Contact AccuSoft Corporation for more information on the status of this tool if your application requires it.

A compiled workspace is created and updated within VisiQuest Visual Programming Environment. First, the visual program is assembled. Then, the glyphs that will make up the compiled workspace are selected; these may include some or all of the glyphs in the visual program. The compiled workspace creation process is then used to automatically generate the first version of the compiled workspace, save it in a toolbox, compile and install the binary.

When a compiled workspace is created or updated, VisiQuest Visual Programming Environment produces two files and passes them to the workspace compiler: the VisiQuest Visual Programming Environment workspace file and the UIS file. The saved VisiQuest Visual Programming Environment workspace file stores the network of glyphs as they appeared in the VisiQuest Visual Programming Environment workspace when the compiled workspace was created. The UIS file defines the GUI and the CLUI of the compiled workspace; it is used to match selections on the compiled workspace glyph's pane with inputs and outputs from the glyphs in the compiled network, as well as any additional parameters that may be set by the user and utilized within the network to affect the run of the visual program.

After the initial creation of the compiled workspace, its graphical user interface and the glyphs in its network may be modified as desired. After each modification, changes to the compiled workspace must be explicitly saved. The workspace compiler will be re-executed with the new UIS file and the new VisiQuest Visual Programming Environment workspace file; it will automatically re-generate, re-compile and re-install the compiled workspace. Note that changes to the GUI involving addition or deletion of parameters or variable name changes will require any saved VisiQuest Visual Programming Environment workspaces that contain the compiled workspace glyph to be updated (this is true for any VisiQuest program that is used as a glyph in a saved VisiQuest Visual Programming Environment workspace).

## Workspace Compiler Types

The VisiQuest workspace file defines the contents of the compiled workspace; its UIS defines its graphical user interface and its command line user interface. Naturally, the structure, behavior, features and limitations of the compiled workspace depend on the workspace compiler that was used to generate it.

VisiQuest Visual Programming Environment, using the flexible code generation system of VisiQuest, is designed to support a variety of workspace compilers. VisiQuest contains two different workspace compilers.

- The Default Workspace Compiler is the standard workspace compiler for VisiQuest. It produces a kroutine which is compiled into a binary.
- The Script Workspace Compiler a sh shell script. It supports only simple sequential visual networks that contain no loops or control structures.

### The Default Workspace Compiler

A workspace compiled using the Default Workspace Compiler is generated as a software object similar to a kroutine. It is compiled into a binary and installed. When the compiled workspace is executed, the same scheduler which is used by VisiQuest Visual Programming Environment to schedule glyphs is used independently to schedule and execute the binaries according to the flow represented by the saved VisiQuest Visual Programming Environment workspace. The compiled workspace now depends only on the binaries of the glyphs used within it. The overhead associated with the VisiQuest Visual Programming Environment visual program itself, as well as the display and update of glyphs in the workspace, is eliminated.

The source code of the compiled workspace is completely auto-generated in the $TOOLBOX/objects/kroutine/oname/src/ directory. No hand editing of the source code should be done.

The main functionality for the compiled workspace is generated in a routine called run_{oname}_compiled_workspace() in the "oname.c" file. This routine opens the VisiQuest Visual Programming Environment workspace file that is saved as part of the software object, and uses calls to the klang library (a private VisiQuest library that implements the VisiQuest Visual Programming Environment workspace scheduler, among other things) to execute the operators specified.

### The Script Workspace Compiler

A workspace compiled using the Script Workspace Compiler is generated as a software object similar to a standard script object. The script for the compiled workspace is completely auto-generated in the $TOOLBOX/objects/script/oname/src/ directory. No hand editing of the generated script should be done.

The generated script contains 'sh' script instructions to execute the binaries represented by the glyphs in the visual network in the order indicated. The Script workspace compiler has only limited scheduling capabilities. It cannot handle looping, conditional constructs such as if/else glyphs, or continuously running glyphs at this time.

## *Creation of Compiled Workspaces*

A compiled workspace may be created and maintained within VisiQuest Visual Programming Environment throughout its life cycle, although the software development tools can also be used on it (with some restrictions that we be mentioned). The main operations involved in creating and maintaining a compiled workspace are:

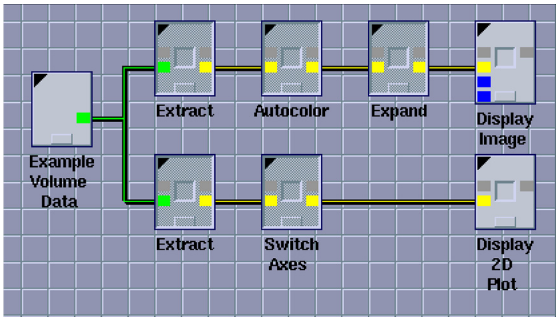
- Creating the visual program
- Creating the compiled workspace from the visual program

- Verifying the correctness of the compiled workspace
- Modifying the graphical user interface of the compiled workspace
- Modifying the visual network contained in the compiled workspace

These operations are explained in detail in the following sections.

## Creating the Visual Program

Use VisiQuest Visual Programming Environment to create or restore the visual program that you wish to make into a compiled workspace. Select the glyphs to be contained in the compiled workspace. Remember not to include the first input glyph(s) or the last output glyph(s) in the network unless you want the input(s) and/or output(s) of the compiled workspace to be hardwired.

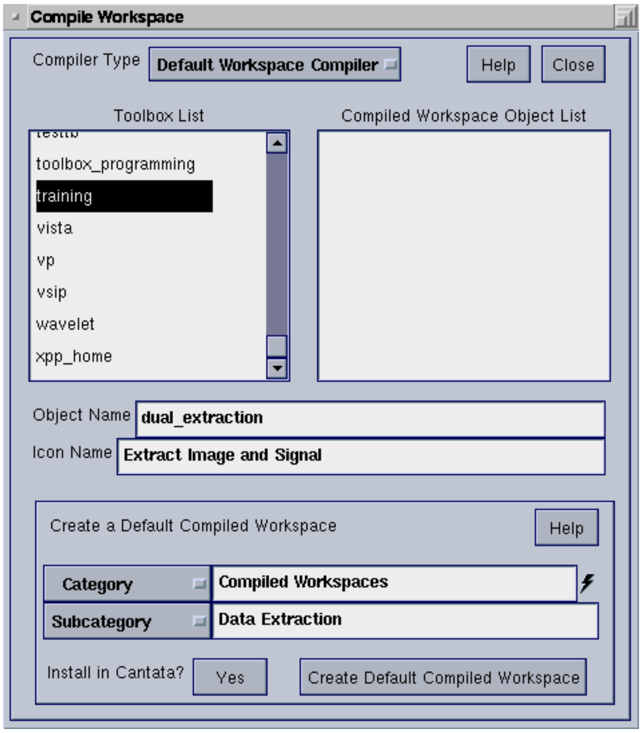


In VisiQuest Visual Programming Environment, the glyphs that will become the compiled workspace are selected.

## Creating the Compiled Workspace

Select "Compiled Workspace" from the Workspace menu. This will display the Compile Workspace pane. A list at the top of the pane allows you to specify the Default Workspace Compiler or the Script Workspace Compiler.

Next, From the Toolbox List, select the toolbox in which the compiled workspace object is to be created. Then, provide an object name, icon name, category, and subcategory. Note that it is wise to set standards with respect to Category and Subcategory, and to re-use these as appropriate. Selecting an existing compiled workspace from the Compiled Workspace Object List will set the Category, Subcategory, and Icon Name to the values for that software object; this allows you to re-use Category and Subcategory names conveniently.



Object Name: dual_extraction

Icon Name: Extract Image and Signal

Category: Compiled Workspaces

Subcategory: Data Extraction

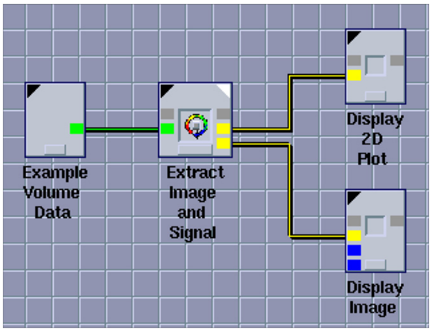
The Compile Workspace pane is available from the Workspace menu.

Set "Install in VisiQuest Visual Programming Environment?" to Yes if you want the compiled workspace to be accessible via the Glyph Menus. Setting this option to No will prevent it from appearing in VisiQuest Visual Programming Environment. Note that you will not be able to modify the GUI or the network within the compiled workspace unless "Install in VisiQuest Visual Programming Environment?" is set to TRUE, since these operations are done from within VisiQuest Visual Programming Environment.

Click on the "Create Compiled Workspace" button to generate, compile, and install the compiled software object in the specified toolbox. VisiQuest Visual Programming Environment's console will report the progress of the compiled workspace creation process. Examine this window to verify that the compiled workspace has been successfully created, compiled, and installed.

**NOTE:** Do not attempt to perform any other VisiQuest Visual Programming Environment operations while the compiled workspace is being created. An information pop-up will appear when the compiled workspace generation process is complete.

After the compiled workspace has been generated, you MUST position the mouse in the VisiQuest Visual Programming Environment canvas, and place the new compiled workspace glyph at the desired location.  The compiled workspace glyph will replace the set of glyphs that were used to create the compiled workspace.



A new glyph is created to represent the compiled portion of the visual program. The little white triangle icon at the upper right is used to open the compiled workspace and see the network contained within.

## Verifying Correctness

At this point, the workspace containing the compiled workspace glyph can be executed normally.  Resetting and running the workspace should produce exactly the same results as the visual program did before it was compiled.

Assuming the "Install in VisiQuest Visual Programming Environment" option was set to Yes during creation, the compiled workspace now will be available for use in VisiQuest Visual Programming Environment. Use the VisiQuest Visual Programming Environment toolbox menus to create an experimental new compiled workspace glyph.

The compiled workspace glyph is similar to a Procedure glyph in that it has the little black triangle at the upper left, and the little white triangle at the upper right. As with the procedure glyph, the little black triangle icon is used to open the GUI of the compiled workspace, and the little white triangle icon is used to display the workspace that was compiled.

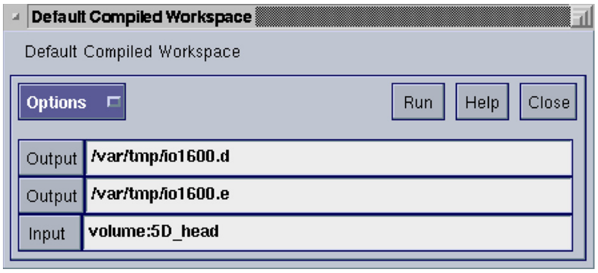
## Modifying the Graphical User Interface

Now that the compiled workspace has been created, you will probably want to adjust the graphical user interface / command line user interface. By default, the GUI will contain Inputfile and Outputfile selections according to the connections of the visual network as they were when the compiled workspace was created. However, they will probably not be in the desired order or position. You may also want to change titles and descriptions to be more self explanatory, and to give appropriate variable names and default values.  You may also export other parameters from the panes of the glyphs within the network to the compiled workspace's GUI, such as integers, floats, flags, and logicals.

Modifications to the GUI of the compiled workspace must be made using the following process:

1. Display the GUI of the compiled workspace by clicking on the little black triangle at the upper left of the compiled workspace glyph.
2. Put the GUI in edit mode by selecting "GUI Editing <Off>" from the Options menu.
3. Display the menuform of the GUI selection you wish to change by clicking the middle mouse button on it.
4. Use the menuform to change the title, default, variable name, etc. as desired.
5. When all selections on the GUI are specified as desired, select "Save Changes" from the Options menu. This will re-generate, re-compile, and re-install the compiled workspace with the modified GUI.

**NOTE:** After changing the GUI of the compiled workspace, never forget to regenerate the compiled workspace by selecting "Save Changes" from the Options menu! If you forget this critical last step, all changes to the GUI will be lost!



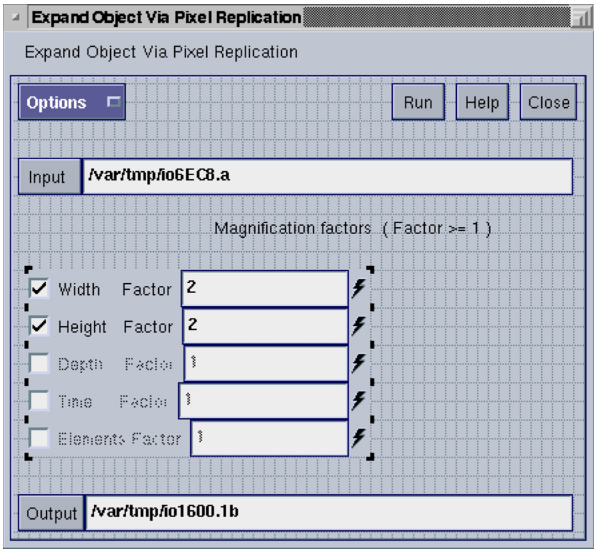
Output /var/tmp/io1600.d
Output /var/tmp/io1600.e
Input volume:5D_head

The compiled workspace's GUI as created by default.

To add additional parameters to the compiled workspace's GUI and CLUI, you may take parameters from glyphs in the compiled workspace and "export" them to the GUI. First, open the compiled workspace glyph and display the compiled network. Then, use the following procedure to "export" parameters:

1. Open the pane of the glyph containing the desired parameters
2. Put the pane in edit mode by turning "GUI Editing <Off>" on the Options menu of the glyph's pane to "GUI Editing <On>."

3. Select the parameter(s) you want to export; when a parameter is selected, it will have control brackets on the 4 corners.


Expand Object Via Pixel Replication
Expand Object Via Pixel Replication
Options
Run
Help
Close
Input
Magnification factors ( Factor >= 1 )
Output

Input: /var/tmp/io6EC8.a

| | | |
|---|---|---|
| ✓ | Width Factor | 2 |
| ✓ | Height Factor | 2 |
| | Depth Factor | 1 |
| | Time Factor | 1 |
| | Elements Factor | 1 |

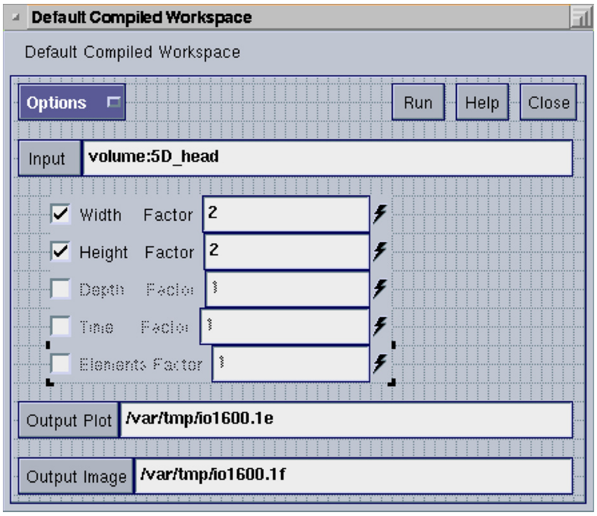
Output: /var/tmp/io1600.1b

The pane for the "Expand" glyph is opened, and put in edit mode. The parameters that specify the magnification factors for width, height, depth, time, and elements are selected for export to the workspace GUI.

4. Finally, export the parameter to the compiled workspace's GUI by selecting "Export to Workspace GUI" from the Options pulldown menu. Note that Export to Workspace GUI" will not be activated in the Options menu unless the pane has been put in edit mode. The parameter you exported should now appear on the GUI.

Repeat the selection exportation process (steps 1-4, above) until all desired arguments from glyphs in the network appear on the workspace GUI.

You may make changes to the titles, descriptions, default values, variable names, etc. of the newly exported parameters by following the process described at the beginning of this section.



Input: volume:5D_head

- [✓] Width Factor: 2
- [✓] Height Factor: 2
- [ ] Depth Factor: 1
- [ ] Time Factor: 1
- [ ] Elements Factor: 1

Output Plot: /var/tmp/io1600.1e

Output Image: /var/tmp/io1600.1f

The compiled workspace’s GUI after exporting width, height, depth, time, and elements parameters from the "Expand" glyph, rearranging selections, and re-naming the two outputfile parameters.

**NOTE:** Do NOT use Guise modify the compiled workspaces' GUI. GUI modifications must be done in VisiQuest Visual Programming Environment using the method described above.

## Modifying the Visual Network

If you would like to modify the visual network that has been compiled, you may do so by opening the workspace of the compiled workspace glyph and making changes as desired.  After changes have been made, the compiled workspace must be re-generated. To regenerate the compiled workspace, select the "Compile Workspace" icon from the Command Bar of the compiled workspace.

The "Compile Workspace" icon only appears in the Command Bar of compiled workspaces. Clicking on this icon will re-generate the compiled workspace; it is the same as selecting "Save Changes" from the Options menu of the compiled workspace's pane.

Note that the "Compile Workspace" glyph only appears on the commandbar of the compiled workspace, not on the main workspace; thus, you will not see it if you close the compiled workspace glyph. You can also regenerate the compiled workspace by selecting "Save Changes" from the Options menu of the compiled workspace's GUI.

**NOTE:** After changing the visual network of the compiled workspace, you must regenerate the compiled workspace! If you forget, changes to the network in the compiled workspace will be lost!

### Executing the Compiled Workspace from the Command Line

Since compiled workspace are generated, compiled and installed as part of the creation process, an compiled workspace may be run from the command line as soon is it is created.

If you have installed the compiled workspace in a toolbox for which you already have the bin directory in your path, just execute:

```
    % rehash
```

Note that if you have created a brand new toolbox to contain the compiled workspace, the new toolbox's bin/ directory may not be in your path.

```
    % set path =($TOOLBOX/bin $path)
```

At this point, you should be able to execute the compiled workspace according to the user interface that you defined for it above. If you used the default workspace compiler, you can check its command line user interface using

```
    % {oname} -U
```

To execute the default compiled workspace using its GUI, use

```
    % {oname} -gui
```

The script workspace compiler is limited in that it does not support the standard VisiQuest options, so [-U], [-gui], [-V], etc, will not work for script compiled workspaces.

### Documentation And Maintenance

As with other VisiQuest objects, documentation for the compiled workspace is added in the html page; the online help page and the man page are automatically generated from the contents of the html page by selecting "Generate Code" from the Make menu of the Commands window in composer, or by executing

```
    % kmake regen
```

in the $TOOLBOX/objects/kroutine/oname/src or $TOOLBOX/objects/script/oname/src directory for default compiled workspaces or script compiled workspaces, respectively.

Like any other program in VisiQuest, the compiled workspace software object attributes can be edited and maintained with the composer software object editor; however, remember that changes to the user interface and the visual network within the compiled workspace must be done from within VisiQuest Visual Programming Environment.

## Data Transports

Data between glyphs is communicated via a temporary data transport. Each connection between the glyphs represents a unique temporary transport. The transport, by default, will be a "file" transport. However, it may greatly increase the efficiency of a workspace to change the data transport used to stream, memory mapped files, or shared memory.

The transport type of any given connection can be changed by clicking on the connection and displaying the connection menu. Any one of the following data transport types may be selected from the connection menu.

- file

  Standard UNIX File (local transport / permanent storage)
- stream

  Standard Stream (FIFO) (local transport / no permanent storage)
- mmap

  Memory Mapped Files (permanent storage).
- shm

  Shared Memory (local transport / permanent storage).

The data transports supported by VisiQuest Visual Programming Environment, listed above, are a subset of the data transports offered by VisiQuest Operating System services. The following data transports are also supported in VisiQuest:

- pipe:

  Standard Pipes (local transport / no permanent storage) socket
- socket:

  UNIX domain socket transport (local transport / no permanent storage)
- tcpip

  TCP/IP socket transport (network transport / no permanent storage)

However, VisiQuest Visual Programming Environment does not support these data transports because the transports only persists as long as the process which created it exists; as soon as the process terminates, the data transport will also terminate.

Not all transports may be available on your machine. For instance, if the machine currently running VisiQuest Visual Programming Environment does not have support for shared memory, the shm transport will not function. However, "shm" will still appear on the connection menu.

When using distributed computing, processes distributed to remote machines must have the input and output connections stored on a file system common to both the local and remote machines. Additionally, transport types other than "file" will not work to a distributed host.

As mentioned above, the data transport used between two processes may be changed by clicking on the connection between two glyphs, displaying the connection menu, and selecting the desired data transport type. This must be done for each connection, a procedure which can be tedious especially if you have a large workspace and you wish to change the transport type of all the connections it contains.

When the data transport type needs to be changed for the majority of glyphs in a workspace, it is easier to change the default data transport type. Normally, the default data transport type is "file." However, the default can be changed by setting the environment variable KHOROS_TEMPFILE to shm, mmap, or stream. Set the variable to file to return to the original default.

```
%setenv KHOROS_TEMPFILE shm
%setenv KHOROS_TEMPFILE mmap
%setenv KHOROS_TEMPFILE stream
%setenv KHOROS_TEMPFILE file
```

## *Transport Buffering*

There are three types of transport buffering models in VisiQuest. The first is for file based transports, the second is for stream based transports, and the third is for memory based transports.

### File Buffering

File buffering is used by the file transport. With the file transport, the data is persistent. The file data transport reads data from the file into the transport buffer. When writing, when the transport buffer gets full the buffer is written back to the file.

In VisiQuest the file buffering has been extended to deal with this intricacy. So the application is free to call read/write operations in any order. This is implemented within the transport buffer by having a validity region which indicates the validity of the data residing in the transport buffer. This enhancement allows us to to get better buffering speeds than available with Sun's FILE buffering (measured using quantify).

### Stream Buffering

Stream buffering is used by the stream, tcpip, socket and pipe transports. It was developed specifically to address compilications involved with attempting to map stream based transports into file based transports. The first complication is due to the fact that stream based transports are not persistent; therefore, after data is read or written it is lost. The second complication is that streams will block when reading or writing too much data. This complicates transports buffering code attempting to generically interface to a transport. The notion of stream buffering was developed to formalize the interfacing to streams and to address these problems. With stream buffering, as data is read or written the transport buffer will accomodate this by dynamically grow to fit all data read or written to date. Seeking on a stream is accomplished by reading or writing to the desired position. One limitation is that the transport buffer is not paged. For example, when streaming a 100 MB file the buffer will grow to 100 MB and then write the data on close.

To override this behavior, applications must open the transport using KOPEN_STREAM, which indicates that persistent is not desired. This causes stream based transports to write the buffer when filled and discard the buffer on read. However, if an application isn't re-reading the data, re-writing the data, or seeking to a previously position this is much more efficient. Both Polymorphic Data Services and Streaming Data Services set KOPEN_STREAM for writing.

Streaming data services additionally sets KOPEN_STREAM for reading.  For Polymorphic Data Services, reading will be stream buffered when using kpds_open_input_object() on a stream based transport.

### Memory Buffering

Memory buffering is used for memory based transports including Shared Memory (shm) and Memory Mapped Files (mmap).  For these transports, the file buffering mechanism also works. However, file buffering is not geared to taking full advantage of the fact that the entire content of the transport is already available in memory.  Therefore, to take full advantage of memory based transports, the buffering model makes the transport buffer and the transport's memory one and the same.  In this manner, memory buffering the allows the memory based transport to read and write directly into memory.

The memory based transports' read and write methods will only be called when trying to read past the end of the available memory.  It is then up to the memory based transport to either indicate EOF has been reached or resize the memory segment making more memory to buffer data to.  Since the operation of resizing the memory segment is very expensive, both shm and mmap resize the memory segment by multiplying the requested size by 25% and rounding up to the nearest page size.  Stream buffering also resizes the transport buffer rounding up to the nearest page size.

To accomodate the ability to have transport buffers be bigger than the data actually buffered, the transport validity has been split into the actual buffer size vs. validity region within the buffer.

## *Directory for Creating Temporary Files*

When a permanent data transport mechanism is used, such as shared memory, or memory mapping, VisiQuest Visual Programming Environment will need a location in which to create the temporary files associated with that data transport. In addition, VisiQuest Visual Programming Environment occasionally creates temporary files for its own use, such as when it needs to display the values of current variables and expressions being used with the expression parser.  All temporary files are erased on a normal exit from VisiQuest unless specifically saved.

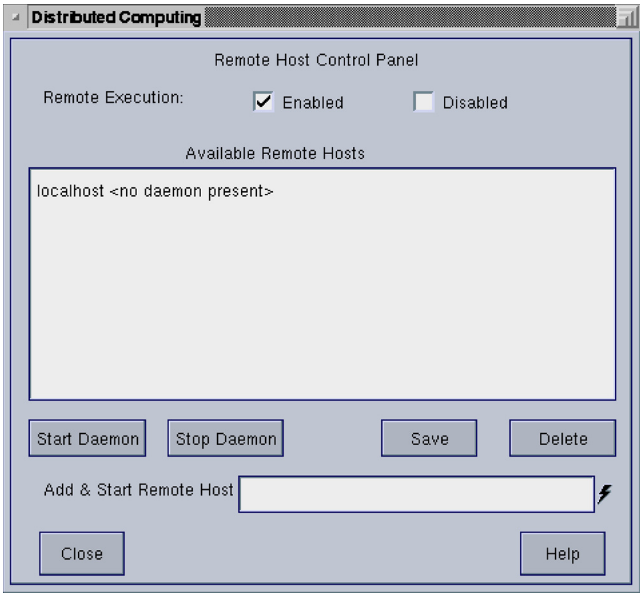
By default, these temporary files will be created in the directory specified by the CWTMPDIR environment variable. The location of this directory may be changed before starting VisiQuest Visual Programming Environment as in the following example command,

```
    CW% setenv TMPDIR /usr/visiquest/tmp.
```

However, this location may be changed as desired while VisiQuest Visual Programming Environment is running by filling in the new location in the Temporary Files text selection. Files created by connecting glyphs prior to changing the Temporary Files directory will NOT be affected by the change.  That is, they will NOT be moved to the new directory.

# Distributed Computing

Distributed computing is available via VisiQuest Visual Programming Environment on UNIX platforms only.  This functionality allows the user to distribute various glyphs to run on remote machines or hosts.



Remote Execution: [✓ Enabled] Disabled

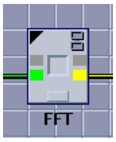
localhost <no daemon present>

Display the Remote Hosts Control Panel by selecting "Configure Remote Hosts" from the Workspace menu of your UNIX installation.

## *Using the Remote Hosts Control Panel*

The Remote Hosts Control Panel is the VisiQuest Visual Programming Environment user interface to the VisiQuest distributed computing capability. Select "Configure Remote Hosts" from the Workspace menu to display the Remote Hosts Control Panel. To enable distributed computing, change the "Remote Execution" toggle from "Disabled" to "Enabled".

When Remote Execution is enabled, the glyphs in the workspace display the remote host icon at the upper right, as shown with the FFT glyph, below.  Also, only when Remote Execution is enabled will the other selections on the Remote Hosts Control Panel be active.



Distributed computing works by dispatching a daemon on each remote host, with that daemon handling all remote process dispatch and communicating process completion back to the local VisiQuest Visual Programming Environment.

The Remote Hosts Control Panel features a list of configurable machines under Available Remote Hosts. If there are no networked machines available for distributed computing, there will be only one entry in the list that reads "localhost <no daemon present>".

For networked machines to be automatically included in the Available Remote Hosts list, they must be specified in the $HOME/.kri/KP2001/khoros_hosts file. This file may be created by hand or with the Remote Hosts Control Panel. The following example shows the syntax used to specify a remote host:

```
tucumcari:oasis:rsh
brandy:mirage:rsh
water:kp2001:rsh
cabernet:guest:rsh
```

Names in the list consist of three parts separated from each other by a colon. For example, the second name in the list, above, is

```
brandy:mirage:rsh
```

In this name, brandy is the name of the remote machine. After the machine name, and separated from it by a colon, comes the log-in name you wish to run under, in this case, mirage. Finally, following a second colon, is the shell for the remote machine; rsh is used here, but any available remote shell may be used, such as ssh or remsh.

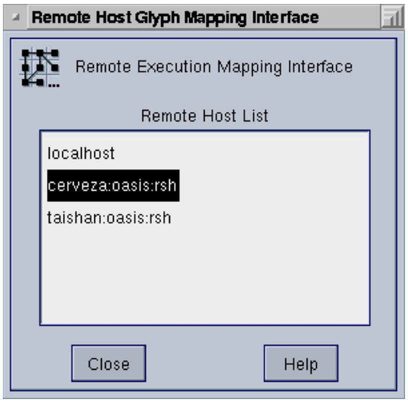
Remote hosts can be added from the Remote Hosts Control Panel. Type the new name into the text box labeled "Add & Start Remote Host" and press return to add a machine to the list. Be sure to utilize the syntax specified above (machine_name:user_name:shell_name). The new remote host will appear in the "Available Remote Hosts" list, and a distributed computing daemon will automatically be started on that machine.

To make machines in the list available for remote execution, use the mouse to select it from the list, then click the "Start Daemon" button. Only those machines with a running daemon will be available for remote execution. The message following the machine name indicates whether the machine has a running daemon or not. Clicking "Stop Daemon" will halt the daemon on the selected machine. "Delete" removes a selected machine from the Remote Hosts list. "Save" saves the machine names in the "Available Remote Hosts" list to the $HOME/.kri/KP2001/khoros_hosts file. "Close" dismisses the Control Panel pane.

## *Dispatching a Glyph to a Remote Machine*

When distributed computing is enabled and remote daemons have been activated on the desired remote machines, glyphs can be dispatched to those machines by clicking on the remote host icon that appears near the top of the glyph. This will pop up a list of all the remote hosts currently available for distributed processing. Remember, only the machines with running daemons will appear in this list.

Selecting a host from the list and then closing the list will dictate to the glyph that it should be dispatched to run on that local machine for all subsequent executions. The host "localhost" is the default and dictates that the glyph will be run locally.


Remote Host Glyph Mapping Interface
Remote Execution Mapping Interface
Remote Host List
Close
Help

localhost
cerveza:oasis:rsh
taishan:oasis:rsh

Clicking on the glyph's remote host icon displays the Remote Host List. Select the desired host from the Remote Host list.

To check which host a glyph is assigned to, position the cursor over the remote host icon.  The status window will display the currently selected host for that glyph.

### *Restrictions*

When using distributed computing, processes distributed to remote machines must have the input and output connections stored on a file system common to both the local and remote machines. If this is not the case, glyphs will produce errors indicating that they cannot read the temporary files produced from upstream glyphs.

Note that a distributed interprocess communication is not yet in place for distributed computing.  Glyphs running on a remote host are scheduled very simply, with process completion indicating that all output data is available and all downstream processes can be scheduled.

## Parallel Host Management

Parallel host management is only available on UNIX platforms if the PARALLEL toolbox is installed.  If this toolbox is not installed, then this selection will be disabled.  There are four control panes available for Parallel Host Management:

- Hosts
- Groups
- Daemons
- Batch Queue

## Hosts

The Hosts control pane provides a list of all known hosts available for parallel processing. New hosts can be added and existing hosts can be removed from this interface. The following information is listed for each machine in the Hosts list.

- Hostname - The fully qualified name of the machine by which it is known on the network.
- Config - The configuration name provided when installing VisiQuest. For homogeneous networks, all architectures will probably have the same config name. The config name is generally different only for different VisiQuest installations on different architectures or operating systems.
- CPUs - The number of processors available on this host. Some HPC machines and symmetric multi-processor (SMP) machines will contain multiple CPUs.
- MPI - The implementation of MPI against which this architecture was compiled. Since different implementations of MPI can not intercommunicate, only hosts compiled with the same version of MPI as the local machine may be used for parallel processing. The hosts which match the local version of MPI are indicated with a >.
- Username - The username used to rsh to the host. By default, this will match your username on your local machine.

The Remove Host button will remove the highlighted entry in the host list. New hosts can be added by pressing the Add Host button after typing in the name of a new host. If a different user name should be used for that host, the Username field can be filled out. The Password field is not yet supported, so you must have permission to rsh to any machine you wish to add. Your remote account should contain a CW.rhosts entry with your local machine and username. The pecho command on the remote machine is used to determine the new host information, so your path on theyremote machine must include the PARALLEL toolbox binaries.

The CW$USER/.kri/KP2001/hosts file is used to store the host information. A different file can be specifying by setting the KHOROS_HOSTS environment variable.

## Groups

The Groups control pane provides a mechanism for specifying directed groups of hosts for execution. A group can be specified for a parallel glyph in VisiQuest Visual Programming Environment, indicating that that parallel operator should be run only on the machines included within that group. Groups can be created or deleted from this interface.

The Group button will pop up a pull-down list of all available groups. The text box next to this pull down indicates the name of the currently selected group. To create a new group, type in a new name into this box and press the Create button. To delete a group, simply press the Delete button. The members of the currently selected group are listed in the center of this control pane. Hosts can be added or deleted from this group using the Add Host and Delete Host buttons. Only the hosts in the Host list which match the local MPI version are available for inclusion in a group.

The CW$USER/.kri/KP2001/groups directory is used to store the group information. Each group list is stored in a separate file. A different group directory can be specifying by setting the KHOROS_GROUPS environment variable.

## Daemons

The Daemons control pane provides control over the daemons in a daemon-based implementation of MPI. Currently, on the LAM MPI is supported from this interface. If your local version of MPI is not LAM, this interface will be disabled.

Daemon-based implementations of MPI require running daemons for process dispatch and communication. This is generally done to reduce process startup time; the time-consuming rsh process is only done once to start the daemons. New parallel processes are then spawned quickly because they are started by the daemons.

The Start Daemons button will start the daemons and the Stop Daemons button will stop the daemons. The current state of the daemons is indicated by a status line above these buttons. Addition LAM-specific functionality is also present, such as the ability to start LAM in a fault-tolerant mode. The Check MPI Process Status button will print out the status to the tty or VisiQuest Visual Programming Environment console. The Kill All MPI Processes button will kill all running MPI processes.

Note that the LAM binaries must be in your path as sourced by your CWcshrc file for the daemons to start. If LAM is installed with the PARALLEL toolbox, the following command will add the proper path:

```
set path = ( `kecho -tb parallel -echo path`/thirdparty/lam61/
bin $path )
```

Note that this must be done after the location of kecho has already been added to your path.

## Batch Queue

The Batch Queue control pane is not currently enabled. This pane will one day allow you to create batch-specific groups which can be used to submit individual parallel programs to a batch system such as LoadLeveler on the IBM SP2.

# VisiQuest Operators

## Introduction

The following table contains listings of the VisiQuest Toolbar toolbox operators that are available via VisiQuest. Table entries are organized alphabetically by Category, Subcategory, and Operator name (icon name), in that order. A brief description of each operator is given, followed by binary name of the program which can be executed from the command line.

## Table of VisiQuest Operators

| **Category** **Subcategory** | **Operator** | **Description** | **Executable** |
|---|---|---|---|
| **Arithmetic** Bitwise Operators | AND | Output = Input 1 AND (Input 2 or Constant) | kbitand |
| | AND Inverted | Output = NOT(Input 1) AND (Input 2 or Constant) | kbitandinv |
| | AND Reverse | Output = Input 1 AND NOT(Input 2 or Constant) | kbitandrev |
| | CLEAR | Set All Bits to Zero | kbitclear |
| | Hadamard | Compute 2-D Fast Hadamard Transform | ifht |
| | Left Shift | Bitwise LEFT SHIFT of Input 1 by (Input 2 or Constant) Bits | kbitlshift |
| | NAND | Output = NOT(Input 1) OR NOT(Input 2 or Constant) | kbitnand |
| | NOR | Output = NOT(Input 1) AND NOT(Input 2 or Constant) | kbitnor |
| | NOT | Perform Bitwise NOT (Invert) Operation on Input | kbitnot |
| | OR | Output = Input 1 OR (Input 2 or Constant) | kbitor |
| | OR Inverted | Output = NOT(Input 1) OR (Input 2 or Constant) | kbitorinv |
| | OR Reverse | Output = Input 1 OR NOT(Input 2 or Constant) | kbitorrev |
| | Right Shift | Bitwise RIGHT SHIFT of Input 1 by (Input 2 or Constant) Bits | kbitrshift |
| | SET | Set All Bits to One | kbitset |
| | XOR | Output = Input 1 XOR (Input 2 or Constant) | kbitxor |

| | | | |
|---|---|---|---|
| **Arithmetic** Comparison Operators | != | IF Input 1 is Not Equal to (Input 2 or Constant), Output = TRUE | kne |
| | < | IF Input 1 is Less Than (Input 2 or Constant), Output = TRUE | klt |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
| | <= | IF Input 1 is Less or Equal to (Input 2 or Constant), Output = TRUE | kle |
| | == | IF Input 1 is Equal to (Input 2 or Constant), Output = TRUE | keq |
| | > | IF Input 1 is Greater Than (Input 2 or Constant), Output = TRUE | kgt |
| | >= | IF Input1 is Greater or Equal to (Input2 or Constant), Output = TRUE | kge |
| **Arithmetic** Complex Operators | Complex to Real | Output = Real, Imaginary, Phase, or Magnitude Component of Input | kcmplx2real |
| | Conjugate | Output(real, imaginary) = Input(real, -imaginary) | kconj |
| | Imaginary Part | Output = Imaginary Component of (Complex) Input | kimagpart |
| | Magnitudes | Output is a Function of the Magnitude of the Input | kmag |
| | Phase | Output = Phase Component of Input [atan2(imaginary, real)] | kphase |
| | Polar to Rect | Convert (r, theta) to (real, imaginary) Coordinates | kpol2rect |
| | Real Part | Output = Real Component of (Complex) Input | krealpart |
| | Real to Complex | Output = Input 1 + j(Input 2) | kreal2cmplx |
| | Rect to Polar | Convert (real, imaginary) to (r, theta) Coordinates | krect2pol |
| **Arithmetic** Linear Transforms | FFT | Fast Fourier Transform (Forward and Inverse) | kfft |
| | LinearOp | Performs linear operations (convolution/correlation) | klinearop |
| **Arithmetic** NonLinear Functions | Bessel | Compute Bessel Functions | kbessel |
| | Error Function | Compute Error & Complement Error Functions | kerf |
| | Exponential | Compute Exponential (antilog) | kexp |
| | Log Gamma | Compute Log Gamma Function | kgamma |
| | Logarithm | Compute Logarithm | klog |

|  |  |  |  |
|---|---|---|---|
|  | ldexp | Output = Input 1 * 2**(Input 2 or Constant) | kldexp |
| **Arithmetic** Single Operand Arithmetic | Absolute Value | Output = Absolute Value of Input | kabs |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
|  | Ceiling | Output = Integer Ceiling of Input | kceiling |
|  | Cube | Output = Input**3 | kcube |
|  | Cube Root | Output = Cube Root of Input | kcbrt |
|  | Floor | Output = Integer Floor of Input | kfloor |
|  | Fractional Part | Output = Fractional Part of Input | kfraction |
|  | Negative | Output = 0.0 - Input | kneg |
|  | Reciprocal | Output = 1/Input | krecip |
|  | Square | Output = Input**2 | ksqr |
|  | Square Root | Output = Square Root of Input | ksqrt |
|  | Truncate | Output = Integer Truncate of Input | ktruncate |
| **Arithmetic** Trigonometry | Atan | Output = Arc Tangent[Input 1 / (Input 2 or Constant)] | katan |
|  | Atan2 | Output = Arc Tangent[Input 1 / (Input 2 or Constant)] | katan2 |
|  | Cos & ArcCos | Compute Cosine or Arc Cosine Function | kcos |
|  | Cosh & ArcCosh | Compute Hyperbolic Cosine or Arc Cosine Function | kcosh |
|  | Degree to Radian | Output = Input * (pi/180) | kdeg2rad |
|  | Hypotenuse | Output = sqrt[(Input 1)**2 + (Input 2 or Constant)**2] | khypot |
|  | Radian to Degree | Output = Input * (180/pi) | krad2deg |
|  | Sin & ArcSin | Compute Sine or Arc Sine Function | ksin |
|  | Sinc | Output = sin(Input)/Input | ksinc |
|  | Sinh & ArcSinh | Compute Hyperbolic Sine or Arc Sine Function | ksinh |
|  | Tan & ArcTan | Compute Tangent or Arc Tangent Function | ktan |
|  | Tanh & ArcTanh | Compute Hyperbolic Tangent or Arc Tangent Function | ktanh |

| | | | |
|---|---|---|---|
| **Arithmetic** Two Operand Arithmetic | Absolute Diff | Output = Absolute Value of (Input 1 - (Input 2 or Constant)) | kabsdiff |
| | Add | Output = Input 1 + (Input 2 or Constant) | kadd |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
| | Blend Data | Output = (Input 1 * Constant) + (Input 2 (1 Constant)) | kblend |
| | Divide | Output = Input 1 / (Input 2 or Constant) | kdiv |
| | Divide Into | Output = (Input 2 or Constant) / Input 1 | kdivinto |
| | Expression | Output = Function(Input1, Input2) | kexprn |
| | Gate Data | Based on Gating Input, Output Values Will Be From Input 1 or Input 2 | kgate |
| | Maximum | Returns Higher Value between Input 1 and (Input 2 or Constant) | kmaximum |
| | Minimum | Returns Lower Value between Input 1 and (Input 2 or Constant) | kminimum |
| | Modulus | Output = Remainder of [Input 1 / (Input 2 or Constant)] | kmod |
| | Multiply | Output = Input 1 * (Input 2 or Constant) | kmul |
| | Power | Output = (Input 1) ** (Input 2 or Constant) | kpow |
| | Replace Value | Replace All Occurrences of X with Y | kreplace |
| | Subtract | Output = Input 1 - (Input 2 or Constant) | ksub |
| | Subtract From | Output = (Input 2 or Constant) - Input 1 | ksubfrom |
| **Data Manip** Analysis & Information | Print Stats | Print statistics to VisiQuest variables | kprstats |
| | Print Value | Print Data Value to Parser and/or Concatenate to File | kprval |
| | Statistics | Compute Statistics of Data Object | kstats |
| **Data Manip** Clustering Operators | K-Means | Perform K-Means Clustering | kkmeans |
| **Data Manip** Convolution | Convolve | Simple convolution and correlation | kconvolve |
| **Data Manip** Data Conversion | Cast Input Types | Upconvert All Inputs to the Highest Input Data Type (K1) | vcast |
| | Normalize | Normalize Data Regions Using Minimum & Maximum of Each Region | knormal |

| | | | |
|---|---|---|---|
| **Data Manip** Frequency Filters | InvFilter | Inverse Filtering (Restoration) in Fourier Frequency Domain | kinverse |
| | Wiener | Wiener Filtering (Restoration) in Fourier Frequency Domain | kwiener |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
| **Data Manip** Histogram Operators | Equalize | Perform Histogram Equalization | kheq |
| | Histogram | Compute Histogram for Data Object | khisto |
| | Histogram Ops | Perform Histogram Equalization and Stretching | khistops |
| | Local Enhance | Enhance Image Using the Local Standard Deviation & Mean (K1) | venhance |
| | Stretch | Perform Histogram Stretching | khstr |
| | Window Eq/Str | Stretch or Equalize Histogram Using Overlapping Windows (K1) | vhxray |
| **Data Manip** Introduce Noise | Exponential Noise | Introduce Exponential Noise in Input Object | kexpon |
| | Gaussian Noise | Introduce Gaussian Noise in Input Object | kgauss |
| | Poisson Noise | Introduce Poisson Noise in Input Object | kpoisson |
| | Rayleigh Noise | Introduce Rayleigh Noise in Input Object | krayleigh |
| | Shot Noise | Introduce Shot Noise in Input Object | kshot |
| | Uniform Noise | Introduce Uniform Noise in Input Object | kuniform |
| **Data Manip** Location Operators | Elevate Data | Create Elevation Location Data From Value Plane | kelevation |
| | Transform Location | Rotate, Translate, and Scale Location Data | klocxform |
| **Data Manip** Map Operators | Map Data | Map Value Data Through the Map | kmapdata |
| | Squish Map | Compress Map to One Column by Means of Average, RMS, or MAX | kmsquish |
| **Data Manip** Mask Operators | Unmask Data | Remove Mask from Data Object | kunmask |
| **Data Manip** Object Attributes | Comment Data | Change the Comment on a Data Object | kcomment |
| | Set Attribute | Modify Data Object Attributes | ksetdattr |
| **Data Manip** Reorganize Data | Flip | Reflect Data Along Specified Axes | kflip |

| | | | |
|---|---|---|---|
| | Reorient | Change Orientation of Data on Dimensions | korient |
| | Switch Axes | Reorient Data on the Width, Height, and Depth Axes | kaxis |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
| | Translate | Translate Data in Object | ktranslate |
| | Transpose | Transpose Data across Dimensions | ktranspose |
| **Data Manip** Size & Region Operators | Accumulate Data | Continuously read and accumulate input | kaccum |
| | Expand | Expand Object Via Pixel Replication | kexpand |
| | Extract | Extract Rectangular Region from Object | kextract |
| | Inset | Inset Object 2 into Object 1 | kinset |
| | Iterate Through Data | Interate through data in an input object | kiterate |
| | Pad | Pad Data with a Constant Value | kpad |
| | Resample | Resample Object Data | kresample |
| | SampleLine | Sample a data object along an arbitrary line | ksampline |
| | Shrink | Shrink Object Via Pixel Subsampling | kshrink |
| **Data Manip** Threshold & Clip Operators | Clip Above | Clip Data Values that are Above Specified Cutoff | kclipabove |
| | Clip Below | Clip Data Values that are Below Specified Cutoff | kclipbelow |
| | Clip Inside | Clip Data Values Inside the Specified Range | kclipin |
| | Clip Outside | Clip Data Values Outside of the Specified Range | kclipout |
| | MegaClip | Clip the Range of Values in Data Object | kmegaclip |
| | MegaThresh | Threshold the Range of Data Values in Data Object | kmegathresh |
| | Thresh Above | Threshold Data Values that are Above Specified Cutoff | kthreshabove |
| | Thresh Below | Threshold Data Values that are Below Specified Cutoff | kthreshbelow |
| | Thresh Inside | Threshold Data Values Inside the Specified Range | kthreshin |
| | Thresh Outside | Threshold Data Values Outside of the Specified Range | kthreshout |

| | | | |
|---|---|---|---|
| Data Manip<br>Segment Operators | Remove Selected | Remove Selected Data Segments from Input | krmseg |
| | Insert Selected | Insert Selected Data Segments from Input 1 into Input 2 | kinsertseg |
| **Category**<br>**Subcategory** | **Operator** | **Description** | **Executable** |
| | Copy Selected | Copy Selected Data Segments from Input | kcpseg |
| | Copy One Segment | Copy One Segment Into Value Segment of Output | kcptoval |
| | Copy Value Segment | Copy Value Segment Into One Segment of Output | kcpfromval |
| | Color Legend | Construct Geometry Representing a Color Legend | gcmaplegend |
| | Extents Box | Generate Geometry Representing Extents Around Data | gextents |
| **Geometry**<br>Color | Colorize | Map Value Data Through a Visualization Colormap | gcolorize |
| | Create Visualization Colormap | Create a Visualization Colormap | ggencmap |
| **Geometry**<br>Data Files to Geometry | Color XYZ Lines | Create Polylines from ASCII XYZ data and a Vis Cmap | glinecolor |
| | Import Facet Data | Import FACET data | gimpfacet |
| | Import PDB | Generate geometry from PDB models | pdb2geom |
| | XYZ Lines | Create Polylines from ASCII XYZ Data | glinexyz |
| **Geometry**<br>Geometry Operators | Flip Normals | Flip all Normals in a Geometry Object. | gflipnorms |
| | Make Cylinders | Replace Lines with Tesselated Cylinders | gtubeness |
| | Transform Location Data | geometric transformation of geometry objects. | gtransform |
| **Geometry** | Print Geometry | Print ASCII Report of Geometry Input | gprgeom |
| Information | Print Location Statistics | Create a directed-points geometry object from raw data. | glocstats |
| **Geometry**<br>Region Operators | Grid 2D Data | Compute z=f(x,y) over a user-defined grid given scatter data. | ggridder2d |
| | Interactive Ortho Slicer | Interactive Slicing Tool. | oslicer |

|  |  |  |  |
|---|---|---|---|
|  | Ortho Slice | Extract (N-1)D Region from (N)D Object | kslicer |
|  | Rake | Create a computational rake. | grake |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
| **Geometry** Te xture Operators | Bind Texture to Geometry | Assign a Texture to a Geometry Object | gbindtexture |
|  | Make Texture Map | Create a 2d texture from an image. | gcreate2dtex |
|  | Te xture Resize | First-order resizing of value segment. | gresample |
| **Geometry** Visualization | Cuberille Data Representatio n | Volume Visualization using Geometric Icons. | gcuberille |
|  | Hedgehog | Generate hedgehog icons at each grid node of a vector field. | ghogs |
|  | Isosurface | Generate surface of constant value from 3d data. | gisosurface |
|  | Make Spheres from Data | Produce Spheres Whose Radii are Determined from Data | gspheres |
|  | Octmesh Maker | Create an Octmesh Geometry Object from 3D Data | goctmesh |
|  | Quadmesh Maker | Create a Quadmesh Geometry Object from 2D Data | gquadmesh |
|  | Render | Interactive Geometry Visualization | render |
|  | Streamlines | Compute Streamlines through 3-Component Vector/Flow Data. | gstreamlines |
| **Image Proc** Classification | Compute Cost | Compute Cost (Surface Arc Length) for Image | icost |
|  | Inverse Median Axis | Compute the Median Axis Inverse Transform | igrow |
|  | LRF-classifier | Classify Image Using the Localized Receptive Field Classifier (K1) | lrfclass |
|  | LRF-training | Calculate Weights for Localized Receptive Field Classifier (K1) | lrftrain |
|  | Labeling | Perform Labeling on Multiband or Cluster Image | ilabel |
|  | Medial Axis Transform | Compute the Medial Axis Transform | imediaxis |
|  | Minimum Distance | Minimum Distance Classifier (K1) | vmindis |

| | | | |
|---|---|---|---|
| | Non-Parametric | Perform N-Dimensional Vector Quantization or Classification (K1) | vquant |
| **Category** **Subcategory** | **Operator** | **Description** | **Executable** |
| | Quantization Error | Compute Error between Quantized Image and Original (K1) | vqerr |
| | Shape Analysis | Perform Shape Analysis on Image | ishape |
| | Spatial Analysis | Compute Spatial Features Using NxM Window | ispatial |
| | Weighted Min Dist | Weighted Minimum Distance Detector (K1) | vwmdd |
| **Image Proc** Color Map Display & Manipulation | Composite | Composite Images | icomposite |
| | Convert Colorspace | Convert Image Colorspace | icolorspace |
| | Create Alpha | Create an Alpha Channel | icreate_alpha |
| | Darken | Darken Image | idarken |
| | Dissolve | Disolve Image | idissolve |
| | Linear Convert | Convert Between Tristimulus Color Spaces | vcltrans |
| | Matrix Convert | Linear Color Space Conversion with Specified 3x3 Matrix (K1) | vcmtrans |
| | Opaque | Make Image Opaque | iopaque |
| **Image Proc** Convolution | Convolve | Perform Convolution or Correlation on Image Data | iconvolve |
| **Image Proc** Data Compression | Compress Colors | Compress Color Multiband Image to Single Band with Map | igamut |
| **Image Proc** Feature Extraction | Fractal Analysis | Fractal Feature Extraction (K1) | vfractal |
| | Fractal Dim P(m,L) | Estimate Fractal Dimension of Image Based on P(m,L) (K1) | vpml |
| | Generate Contour Image | Generate Contour Image | icontour |
| | Te xture | Texture Feature Extraction Using LAW Metrics | itexture |
| **Image Proc** Frequency Filter | Band-Pass | 2-Dimensional Band-Pass Filter Design | ibpf |
| | Band-Stop | 2-Dimensional Band-Stop Filter Design | ibsf |

| | | | |
|---|---|---|---|
| | High-Pass | 2-Dimensional High-Pass Filter Design | ihpf |
| | Low-Pass | 2-Dimensional Low-Pass Filter Design | ilpf |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
| **Image Proc** Geometric Operators | Flip Image | Reflect Image Data Along Either Width Or Height Axes | iflip |
| | GeoWarp | Perform Direct Bilinear Geometric Warping | igeowarp |
| | Rotate | Rotate Object by Arbitrary Angle | irotate |
| **Image Proc** Nonlinear Filters | Median | Median Filter for Image Data | imedian |
| **Image Proc** Organize Data | Extract 1 Band | Extract a Specified Data Band from a Multiband Image | ibandsp1 |
| | Extract 3 Bands | Extract 3 Bands from a Multiband Image (K1) | vbandsp3 |
| | Mosaic to Multiband | Convert Mosaic Image to Multiband Image (K1) | vmos2band |
| **Image Proc** Segmentation | Border Distance | Compute Distance from the Boundary in a Tw o-Valued Image | idistance |
| | Color Threshold | Generate a Binary Image by Thresholding a Color Image | igamth |
| | Edge Closing | Close Boundaries of an Edge Image (K1) | vclose |
| **Image Proc** Spatial Filters | DRF Edge Extract | Optimal Difference Recursive Filter for Edge Detection (K1) | vdrf |
| | GEF Edge Extract | First Derivative Operator for Symmetric Exponential Filter (K1) | vgef |
| | Gradient Operator | Gradient Operators (Roberts, Sobel, Prewitt, Isotropic) | igradient |
| | Median Histogram | Median Filter the Image Using Histogram to Find Median (K1) | vhmed |
| | SDEF Edge Extract | Second Derivative Operator for Symmetric Exponential Filter (K1) | vsdef |
| | Speckle Removal | Reduce Speckle Noise Using the Crimmins Algorithm (K1) | vspeckle |
| **Image Proc** Surface | Add Tilt | Add a Specified Illumination Gradient Plane to Image (K1) | vtilt |
| | Find Tilt | Compute the Best-Fit Plane Parameters for Input File (K1) | vgettilt |

|  |  |  |  |
|---|---|---|---|
|  | Normals and Angles | Compute Surface Parameters (Normals and Angles) (K1) | vsurf |
|  | Remove Tilt | Remove Illumination Gradient by Subtracting Best-Fit Plane (K1) | vdetilt |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
|  | Slopes and Aspects | Compute Slope and Aspect Images from Elevation Data (K1) | vslope |
| **Input/Output** Conversion | KDF To Stream | Convert from polymorphic to streaming | pds2ds |
|  | Stream To KDF | Convert from streaming to polymorphic | ds2pds |
| **Input/Output** Data Files | Animation Data | Image Sequences Over Time and/or Depth | sequences |
|  | Example Volume Data | Volumetric Data Sets | volumes |
|  | Example Volume Data | Volumetric Data Sets | volumes |
|  | Images (Misc) | Sample Image Data | images |
|  | RGB Images | Sample RGB Color Images | rgbimages |
|  | Signals (1D) | 1D Signals or Sequences | signals |
|  | Surface (3D Plot) | Two Dimensional data for 3D Plotting | plot3D |
| **Input/Output** Generate Data | 2D Gaussian | Generate Gaussian Function Image | igauss_func |
|  | Box Projection | Create Image of Parallelogram Projected onto CCD Sensor | igbox |
|  | Circle Image | Create Image of a Filled Circle | igcirc |
|  | Constant | Generate Object Containing Constant Value Data | kgconst |
|  | Fractal Image | Create Fractal Image with Specified Fractal Dimension (K1) | vgfractal |
|  | Generate Expression | Generate Object Value Data From Expression | kgexprn |
|  | Generate Location | Generate Location Data | kgenloc |
|  | Impulse | Generate Object Containing Impulse Value Data | kimpulse |
|  | Marr Filter | Create a Marr Type Edge Detection Filter Kernel (K1) | vmarr |

|  |  |  |  |
|---|---|---|---|
|  | Piecewise Linear | Create 2D Piecewise Linear Periodic Function (K1) | vgpwl |
|  | Sinusoid | Generate Object Containing Sinusoidal Value Data | kgsin |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
| **Input/Output** Generate Noise | Exponential Noise | Generate Exponential Noise Data | kgexpon |
|  | Gaussian Noise | Generate Gaussian Noise Data | kggauss |
|  | Poisson Noise | Generate Poisson Noise Data | kgpoisson |
|  | Rayleigh Noise | Generate Rayleigh Noise Data | kgrayleigh |
|  | Uniform Noise | Generate Uniform Noise Data | kguniform |
| **Input/Output** Hardcopy Output | Postscript | Convert Image to Postscript | ipostscr |
| **Input/Output** Information | Data Object Info | Print File Information | kfileinfo |
|  | File Viewer | Display online help (or ascii text files) | khelp |
|  | Print Data | Print Data in Formatted ASCII | kprdata |
| **Matrix** Arithmetic | Invert Diagonal | Compute Inverse of Diagonal Matrix | minvdiag |
|  | Invert Matrix | Compute Inverse Matrix | minvert |
|  | Matrix Add | Output = Matrix 1 + Matrix 2 | madd |
|  | Matrix Conjugate | Compute the Complex Conjugate Matrix | mconj |
|  | Matrix Multiply | Matrix Multiply (C=AB) | mmul |
|  | Matrix Power | Compute the Positive Integral Power of a Square Matrix | mpow |
|  | Matrix Subtract | Output = Matrix 1 - Matrix 2 | msub |
|  | Matrix Transform | Transform all vectors in an object using a single matrix | mtransform |
|  | Scalar ABS | Output = Absolute Value of Input (componentwise) | mscalar_abs |
|  | Scalar Add | Output = Input Matrix + Constant | mscalar_add |
|  | Scalar Divide | Output = Input Matrix / Constant | mscalar_div |
|  | Scalar Multiply | Output = Input Matrix * Constant | mscalar_mul |

| | | | |
|---|---|---|---|
| | Scalar Power | Output = Input Raised to a Power (componentwise) | mscalar_pow |
| | Scalar Recip | Output = Reciprocal of Input (componentwise) | mscalar_recip |
| | Scalar Sqrt | Output = Sqrt of Input (componentwise) | mscalar_sqrt |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
| | Scalar Subtract | Output = Input Matrix - Constant | mscalar_sub |
| | Transpose Matrix | Transpose Rows and Columns of Matrix | mtranspose |
| **Matrix** Decomposition | LU Decomposition | Compute LU Decomposition of Matrix (Input = L*U) | mlud |
| | SVD | Compute Singular Value Decomposition (SVD) | msvd |
| **Matrix** EigenValues/Vec-tors | Eigen | Compute Eigenvectors and Eigenvalues of a Square Matrix | meigen |
| **Matrix** Generation | Gen Diag Matrix | Generate a Diagonal Matrix | mgdiag |
| | Identity Matrix | Generate Identity or Unit Matrix | mgident |
| | Null Matrix | Generate Null or Zero Matrix | mgnull |
| **Matrix** Linear Operators | Covariance Matrix | Compute Covariance Matrix | mcovar |
| | Simultaneous Eqns | Least Squares Solution to System of Linear Equations | mlse |
| **Matrix** Utilities | Exchange row/ col | Exchange Rows or Columns of a Matrix | mexchg |
| | Extract Col | Extract a Column from a Matrix | mextract_col |
| | Extract Row | Extract a Row from a Matrix | mextract_row |
| | Extract diag/ row/col | Extract Matrix Diagonal, Row, or Column | mextract_diag |
| | Replicate Submatrix | Replicate a Given Matrix as Submatrices of a Larger Matrix | mreplicate |
| | Row/Col Sums | Compute Row or Column Sums for a Matrix | mrcsum |
| **Program Utilities** General | Command Icon | Command_icon used to run a generic command | command_ico n |
| | Comment Icon | Comment a VisiQuest workspace | comment |
| | Counter Display | Display the Output from sequence counter | countdisplay |

| | | | |
|---|---|---|---|
| | File Sequencer | Utility for sequencing of file display in VisiQuest. | kcat |
| | Generic Interface | Generic Routine Interface (glue) | kgeneric |
| | Loop Files | Loop through file list and output data. | kloopfiles |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
| | Sequencer Counter | Utility for count sequencing in VisiQuest. | kcount |
| | Termination Test Routine | VisiQuest test routine to check exit status | kruntest |
| | Test Continuous Run Routine | A test case for the continuous run proceedure. | kcontrun |
| **Stream** Binary | sabsdiff | Absolute difference of two streams | sabsdiff |
| | sadd | Add two streams | sadd |
| | sarctan2 | Arctangent of second stream divided by the first stream | sarctan2 |
| | sdiv | Divide first stream by second stream | sdiv |
| | sdivinto | Divide second stream by first stream | sdivinto |
| | shypot | Compute the Euclidean distance of two streams from origin | shypot |
| | sldexp | Computes x * 2ˆy | sldexp |
| | smaximum | Maximum of two streams | smaximum |
| | sminimum | Minimum of two streams | sminimum |
| | smodulo | Modulo of two streams | smodulo |
| | smult | Multiply two streams | smult |
| | spow | Computes xˆy | spow |
| | ssub | Subtract second stream from first stream | ssub |
| | ssubfrom | Subtract first stream from second stream | ssubfrom |
| **Stream** Unary | sabs | Absolute value | sabs |
| | sarccos | Arccosine of stream | sarccos |
| | sarccosh | Hyperbolic arccosine of stream | sarccosh |
| | sarcsin | Arcsine of stream | sarcsin |
| | sarcsinh | Hyperbolic arcsine of stream | sarcsinh |

| | | | |
|---|---|---|---|
| | sarctan | Arctangent of stream | sarctan |
| | sarctanh | Hyperbolic arctangent of stream | sarctanh |
| | scbrt | Cube root of stream | scbrt |
| **Category Subcategory** | **Operator** | **Description** | **Executable** |
| | sceil | Ceiling of stream | sceil |
| | scos | Cosine of stream | scos |
| | scosh | Hyperbolic cosine of stream | scosh |
| | serf | Error function of stream | serf |
| | serfc | Complementary error function of stream | serfc |
| | sfloor | Floor of stream | sfloor |
| | sfraction | Fraction of elements of stream | sfraction |
| | sgamma | Log gamma function of stream | sgamma |
| | sneg | Negation of stream | sneg |
| | srecip | Reciprocal of stream | srecip |
| | ssin | Sine of stream | ssin |
| | ssinc | Sinc (sin(x)/x) of stream | ssinc |
| | ssinh | Hyperbolic sine of stream | ssinh |
| | ssqrt | Square root of stream | ssqrt |
| | stan | Tangent of stream | stan |
| | stanh | Hyperbolic tangent of stream | stanh |
| | strunc | Truncation of elements of stream | strunc |
| **Visualization** Color Map Display & Manipulation | Autocolor | Automatically Color Data Object by Manipulating the Color Map | kautocolor |
| | Change Colorspace | Change the color space attribute on a data object | kcolorspace |
| | Display Palette | Non-Interactive Display of Color Palette | putpalette |
| | Interactive Look Up Table | Interactively Edit Colormap using Look Up Table Method | editlut |
| | Interactive Threshold | Interactively Edit Colormap Using Threshold Method | edithresh |
| | Print Color Map | Print Color Map Values Associated with Image Pixel Value | putmapval |

| | | | |
|---|---|---|---|
| | Pseudo Color | Interactively Edit Colormap Using Pseudo-color Method | editpseudo |
| **Visualization**<br>Image Capture | Get Image | Capture image from screen to Khoros 2.1 KDF | getimage |
| **Category**<br>**Subcategory** | **Operator** | **Description** | **Executable** |
| **Visualization**<br>Interactive Image Display | Animate | Interactive Animation of Image Sequence | animate |
| | Edit Image | Interactive Image Display and Manipulation | editimage |
| **Visualization**<br>Interactive ROI Extraction | Extract ROI | Interactive image Region of Interest (ROI) extraction | extractor |
| **Visualization**<br>Non-Interactive Image Display | Display Animation | Non-Interactive Animation of Image Sequence | putanimate |
| | Display Icon | Non-Interactive Display of Iconified Image | puticon |
| | Display Image | Non-Interactive Image Display | putimage |
| | Display Zoomed Image | Non-Interactive Display of a Zoom Image | putzoom |
| | Print Pixels | Display Pixel Values Associated with Region of Image | putpixel |
| **Visualization**<br>Plot Display | Display 2D Plot | Non-Interactive 2D Plot Display | putplot2 |
| | Display 3D Plot | Non-Interactive 3D Plot Display | putplot3 |
| | Interactive 2D/ 3D Plot | Interactive 2D & 3D Plotting | xprism |
| **Visualization**<br>Spectral Analysis | | | |
| | spectre | Multi-spectral Image Display | spectre |