The Giver of Names Manual

Table of Contents

Guide to the Manual

This manual includes:

- Comprehensive descriptions of the installation and its parts
- Drawings of the recommended layout
- Drawings of past exhibitions with comments
- Equipment descriptions
- A connection diagram
- A guide to installing and setting up the software

To aid in the porting of the Giver of Names software to other computer systems in the future, the following highly technical sections are included:

- A guide to porting the software to other Operating Systems
- An introduction to the API for connecting to the Core Giver of Names code
- An introduction to the Max OS X Cocoa implementation
- An introduction to the Core Software components

Physical Setup Description

Space
The installation space should have light gray or white walls and should be a minimum of 6 meters x 6 meters. Ceiling height is not critical, but it is preferable that the ceiling be at least 2.5 meters high.

Arrangement
There is a pedestal wrapped in black velvet towards the back or back corner of the space. Somewhat above the pedestal hangs a small projection screen. Balancing this in the opposite half of the room, a computer hangs on a suspended shelf with its screen facing away form the pedestal. Fairly close to the computer and facing the top of the pedestal, a camera sits on a tripod. A cluster of toys and domestic objects sits on the floor in the space between the camera and the pedestal.

The orientation of the installation should be such that the visitor sees the projection screen and pedestal on entrance to the space. They may also see the computer screen, but the first impression is ideally of the projection screen, and pedestal and anything on the pedestal. The main axis of the piece may on the diagonal or along the longer axis of the room. A diagonal is useful in smaller spaces. It also has the positive effect of increasing the clarity of the shadows cast by the objects on the pedestal by the small pedestal lights.

The computer screen's orientation can be so that the screen looks away from the pedestal or looking to the side, perpendicular to the line form camera to pedestal. Ideally a visitor can position themselves comfortably so that they can see the pedestal and projection and computer screen with a minimum of movement.

The lighting level in the room is quite low and the top of the pedestal is lit so that the objects on the pedestal almost seem to glow. The small lights on the pedestal ideally cast discernable object shadows on the walls to one side and the other of the pedestal. The objects on the floor can be lit, but at a somewhat lower level than the pedestal.

Pedestal
The pedestal should be 1 m (h) x .5 m x .5 m. As there are lights to be mounted inside the pedestal to protrude through holes near the corners, the construction should be probably made with sheets of MDF rather than having an internal frame with thinner cladding, as the frame may block the preferred location for these holes. The pedestal should be bottomless to allow the lights to be mounted inside. (see drawings)

Pedestal Position
The pedestal should be a minimum of 6 feet from any nearby wall but ideally the walls to the left and right behind the pedestal are about the same distance from the pedestal (for shadows).

Velvet

The outside surfaces of the pedestal are covered in black velvet. The velvet should be chosen for darkest blackness. Cotton velvet tends to underperform synthetic velvet for this. In any case, velvet generally has a bias and is much less dark when oriented one way versus another. Before attaching the velvet it is important to place the pedestal in light somewhat similar to the actual lighting in the installation (light from above and to the front.) Then you can accurately choose the correct orientation of the velvet to provide the darkest top and front from the camera's perspective (looking at the pedestal with your eye at 1 meter above the ground.)

Before attaching the velvet, make sure that the holes for the lights have already been drilled. Also make sure that a suitably-sized notch has been cut in the bottom of the back face for the power cable. The front of the pedestal will be the side whose left and right edges are closest to the holes for the lights. The back side has left and right edges farthest from the light holes (and has the notch in the bottom)

The velvet is usually applied in two sections. First, one piece is stretched across the top and stapled to the sides. This piece should be large enough to drape over the sides by about 2 - 4 cm on all sides. Verify the orientation of the velvet for best black viewed from the front at 1 meter before stapling. It is then usually best to cut the excess fabric form the corners, leaving enough overlap at the corners to that the corner tip is not exposed, and tapering the fabric back so that the flap on each side is slightly narrower than the pedestal side, so that there is no excess fabric to unnecessarily bulk up the corners.

Then a single approximately 48" wide piece is attached around the pedestal, with the join at the back. Before wrapping the pedestal, it is important to finalize the orientation of the velvet. Sometimes, especially with high quality synthetic velvet, it is hard to determine the best orientation. The best choice can usually be determined by folding an inch or so of the velvet along what will be the top, and placing it against the pedestal front near the top under the kind of light the pedestal will be in in the installation (from the front and above). In a bad orientation, the fold will pick up excessive light even though the flat face of velvet appears quite black. As the camera sees the top section of the pedestal, it is most important that you minimize the amount of light that the top edge of the fold of velvet reflects.

With the front of the pedestal facing away, lay the corner of the velvet piece face-down on the top of the pedestal, with the 48" dimension running toward the front of the pedestal. Drape 4 - 6 cm of the velvet over the edge facing towards you. Make sure that the overhang of fabric is the same across the facing side of the pedestal, smooth it down and then staple the velvet to the facing side of the pedestal with the staples a few millimeters below the edge of the top. Put at least one staple every 4 cm. Maintain enough tension to keep the velvet from bagging between staples. Be careful not to staple to close to the top of the pedestal, as your staples will miss the wood and the staple teeth will be exposed through the velvet.

Then rotate the pedestal 90 degrees, and repeat the above process for each of the other sides of the pedestal. Make sure you keep an equal amount of overhang all the way round. The velvet may seem a little unwieldy, but as long as you maintain a reasonably amount of tautness and keep the overhang regular, all will work out in the end.

When you have stapled all 4 sides, rotate 90 degrees one more time and repeat the process approximately 1/2 way across the first (back) side. You should now be stapling over the already covered back side. Leave about 10 cm unstapled from the middle of the pedestal back and cut the excess velvet away (cutting or ripping along the 48" dimension (or perpendicular to the staple line of the stapled velvet.

Now you have an inverted sheath, and you can carefully pull it down over the pedestal to sheath the pedestal and expose the black side of the velvet.

Verify that no staples are sticking out of the top.

Flip the pedestal so that the top rests on the floor. Starting with the first-stapled side, wrap the velvet over the bottom edge of the pedestal and pull it taut, careful to keep the velvet on the side smooth. Staple it to the inside of the pedestal in a few places, especially near the corners. Then continue around until you are at the overlapping section at the back face. It is usually best at this time to shift the pedestal to be on its front face. Fold the loose flap of velvet in half so that the cut / ripped edge is folded under. Then arrange the top edge so that it can be folded under to meet the top edge without sticking up over the edge. Staple it at the top through the velvet. (The staple will not be visible as this side of the pedestal is not lit.) Pull the folded edge down to the bottom and arrange it to that you can staple it without leaving the back flap baggy. Then staple along the folded edge in 5 or 10 places. These exposed staples can be hammered in to be less visible.

Using a box-cutter, cut an X in the velvet at the location of the two lighting holes. The lights will protrude through these holes.

Projector

The projector will be mounted to the ceiling (or dropped with a pipe mount) positioned upside-down with the bottom face of the projection just above the top of the projection screen and to the front (towards the computer). It may be possible to position the projector so that it can be flat, but it can also be mounted higher, tilted down and the screen size and shape corrected with keystoning. The decision to mount straight or angled down is a subjective one related to how it feels to have the projector in the space at a particular height. The projector height should be worked out to prevent it from blocking anyone's view of the screen, but not so high as to be beyond the capabilities of the projector to adjust keystoning. Care must be taken when choosing the projector mounting location that the range of the projector's zoom lens allows for adjusting the image size to fit the screen. (with a few millimeters of black around the outsides)

Projectors often have an auto-adjustment feature that attempts to optimize the image for the signal that is being received. In some cases this is welcome, and in others, the fact that the projection scree is often very black means that auto-adjustment fails to provide an appropriate adjustment. It is probably preferable to disable auto-adjust so that the projector does not modify its settings after all the installation adjustments have been made.

Projection Screen

The projection screen hangs over the top of the pedestal wit the bottom of the screen approximately 60 - 70 cm above the top of the pedestal. The screen should hang about 1/3 of the pedestal depth form the back of the pedestal. The size of the screen is 60 cm (w) x 45 cm (h). The screen material can be any robust flat paintable material, with a thickness depending on material (but probably ranging from 1/8" to 1/4".The front surface should be painted approximately 50-60% grey matte. (The optimal darkness of grey depends a bit of projector) The colour should be as neutral as possible.The back surface can match front colour, for simplicity. Two holes or eyelets can be used for hanging. The hanging method should allow for easy leveling (turnbuckle / aircraft cable?) (monofilament is an option but a little harder to level)

Camera

The camera sits near the computer on a low but sturdy tripod. The lens of the camera should be positioned to look right across the top of the pedestal at the height of the pedestal. The camera should not see the top of the pedestal, but also, the the bottom of objects placed on the pedestal should not be occluded by the front edge of the pedestal top. The tripod should be weighted for stability. The camera lens allows for different distances from the pedestal. The preferred distance is between 3 and 4 meters from the camera lens front to the facing side of the pedestal. This dimension should expand and contract in relation to the size of the room. In very large spaces, it might be advisable to stretch the distance above 4 meters, but it is a mistake to make the installation try to fill a huge room. The camera field of view should be adjusted so that the scale of the objects seen by the camera as seen in the projection is as close as possible to the scale of the actual physical objects. They should also line up vertically.

Computer

The computer should be placed on a suspended shelf, hanging from the ceiling by threaded rod (or otherwise hung from the ceiling. Transparency and lightness are the ideal. It should be approximately 5 meters from the pedestal. The height of the computer screen should be such that a visitor of average height looks straight ahead at the center of the screen (i.e. the center of the screen should be at a height of about 142 cm). Suspending the shelf with aircraft cable is possible, but the computer will tend to swing unless care is taken to stabilize it. In some cases, the shelf has been tied down to the floor to prevent swinging. As the light levels in the installation are usually fairly low, the screen should usually be turned down towards its minimum brightness. The computer’s own speakers are usually used for the voice output, but in some cases, if there is too much competition, a small pair of multimedia speakers will suffice to boost

the volume. The volume should be audible throughout the room but not necessarily understandable throughout the room. It should not be aggressively loud.

The keyboard and mouse are needed to set up the software, but are removed for exhibition.

Connections

(see the connections diagram in Appendix A)

Power

The computer is plugged into power.
The projection is plugged into power.

Video cable to projector

An adaptor called a miniDVI adaptor is plugged into the back of the iMac computer. The other end of the adaptor has a VGA connector. This conected to a VGA cable going to the VGA input of the projector.

Firewire Cable from Camera to Computer

A firewire 400 cable is connected from the firewire 400 plug on the back of the computer to the firewire 400 plug on the camera. In the case where only a firewire 800 plug is available on the computer, a simple adaptor cable that converts from firewire 800 to 400 can be used.

Software

Objects

The objects used in the installation have been chosen for optimal results in the installation. The installation does not require these specific objects. If objects are damaged or lost, they may be replaced with similar objects, or objects that:
- are not shiny or clear (not well read by the camera)
- that have a relatively coherent shape. (iconic shapes are particularly good)
- have substantial areas of solid colour rather than detailed patterns
- can stand on their own on the pedestal.
- are not so dark that they will tend to disappear into the black felt.

The system is particularly good at recognizing certain shapes and it is good to have some things of these shapes in the mix: boots, bowling pins, spheres or circles, cones, squares, hearts, and other things of simple shape. Objects that can stand up by themselves to present their most iconic profile to the camera are desirable. Objects that lie flat are not very effective. Objects with unusual colours are also good. Most things that meet the other requirements tend toward the primary colours, but the system is quite sensitive to a wide range of more complex colours.

The objects are placed in a cluster in the space in between the camera and the pedestal. The cluster should be 2 meters in diameter or less. They should be gathered form time to time to keep them from dissipating into the space. The pile of objects serves a secondary pragmatic purpose which is to discourage people from standing in between the camera and the pedestal. They should be gently lit from above. The lighting on the objects should not give the impression that the toys are 'staged' or that the arrangement is precious, as this will discourage people from picking up the toys and placing them on the pedestal.

Lights and Lighting

The space should not have any natural light. The pedestal will be lit by the two littlites mounted on the pedestal and two highly focusable lights on the ceiling. The lights on the ceiling should be looking down at an angle of approximately 45 degrees and be 45 degrees to the left and right of the pedestal, lighting the pedestal from the camera side. There should also be one soft light flooding the pile of toys with a lower level of light than that on the pedestal.

The ceiling-mounted lights are intended to light the top of the pedestal and not the pedestal front (although some light on the front is inevitable. The lights should be adjusted and framed to have minimal spill onto the projection screen, but still able to light the top of reasonably tall objects on the pedestal.

The pedestal lights should be extended on their goose-necks to sit just outside the camera's field of view to the front left and front right and to have their light bulbs just at or above the pedestal top. The intention is to light the objects as evenly as possible, with as little undershadow as possible, while not washing out the projection. These lights should have the brightness control at maximum. The pedestal lights have the desirable side effect of casting shadows of the objects on the adjacent walls. They can be adjusted to optimize these shadows. For example, by extending the goosenecks to their maximum extension, the cast shadows grow sharper and smaller (which is desirable). The pedestal lights can also be adjusted to reduce the amount of light spilling on to the back wall. Ideally the camera will see the back wall behind the objects as being black. This will be true even if to the eye, the wall seems gray, and it is therefore best adjusted with the computer on, the camera in position and the software showing the live image on the projection. Clear covers are available to protect the bulbs which tend to burn out more quickly if exposed to the oil from fingers.

Screen / Object Balance

In an ideal situation, the image on the projection looks as similar to the real objects on the pedestal as possible. Most importantly, the scale of the objects in reality and in projection must match as closely as possible. Secondly, then general brightness of the projection should approximate that of the objects. Thirdly, the colour temperature should be as close as possible. Note that some objects (particularly purple ones) may not appear to be the right colour in the projection, even though everything else seems well matched. This may be unavoidable.

Brightness of Computer Screen
The computer screen will probably need to have its brightness turned down to be in balance with the rest of the installation. The LCD screen will likely have a dull gray glow in all the black sections, and the brightness level must be carefully chosen to balance the objects and projection on one hand while not getting excessively washed out by the gray glow.

Audience Considerations
Depending on the context, visitors to the installation will display varying levels of hesitancy about picking up objects to place on the pedestal. If the installation is positioned amongst more formal displays, this may reduce their comfort with touching the work. Excessively theatrical lighting on the pile of toys can also imply to the visitor that the objects have been specifically arranged and should not be touched. If it is determined that there is a low likelihood that people with interact with the work in a particular manifestation, it is acceptable to provide some sort or prompting. This could be in the form of a minimal sign or didactic panel, or animators or docents offering suggestions.

Visitors sometimes feel the desire to pile as many objects as possible on the pedestal. This is completely acceptable behaviour, but sometimes diminishes the experience of the next visitor, as the initial discovery of the functioning of the piece is generally more effective if they start with one or two objects on the pedestal. Sometimes, it is useful to suggest to a visitor to try one or two objects initially. It is also a good idea for gallery staff to be in the habit of removing piles of objects from the pedestal if the installation is empty of visitors.

Oversight during Exhibition
During an exhibition of the Giver of Names, there are several things to keep an eye on.

Successful detection of the empty pedestal: When the pedestal is empty, the software should not capture a new image, but should continue describing the previous objects. If the software does start grabbing images of the empty pedestal, then either a significant amount of dust has accumulated on the pedestal, or something in the lighting has changed. The pedestal should be cleaned up and the Black Level should be adjusted using the “Adjust Black Level” menu command.

Proper position of camera: Occasionally, a visitor will knock the camera out of its proper position. The objects on the pedestal may never appear, or may not line up with the images of those objects in the projection. The camera should be repositioned using the technique described for setting up the installation elsewhere in this manual.

Pedestal lights functioning: Very often, the pedestal lights will not be turned on. This adversely affects the operation of the visual recognition software. Most gallery attendants do not notice the absence of the light, so it should be checked daily and noted.

The objects start to wander: At regular intervals, it is desirable to have the objects gathered back together in a rough circle in the space between the pedestal and the camera.

Broken toys that pose a potential hazard should be removed.

Equipment Details

Computer
Apple iMac intel 2.0 gHz processor
minimum 100 gByte HD
minimum 1 gigabyte of RAM
minimum 17” screen

Camera
640 x 480 colour firewire (IIDC) camera with varifocal lens
camera body: The Imaging Source DFK 21AF04 (www.theimagingsource.com)
lens: c-mount, Varifocal lens with minimum range of 8-25 mm, manual iris
recommendation: Pentax C70509 5-50mm manual iris varifocal lens

Projector
minimum resolution: SVGA 800 x 600
minimum brightness: 1500 lumens
DLP projector with 1500:1 contrast or better is desirable (LCD okay)
zoom lens (minimum zoom ratio 1.2) to allow some leeway during install.
MUST HAVE KEYSTONING to allow projector to project down onto screen from somewhat above.
projection size will be small: (60 x 45 cm image onto dark grey painted screen)
source will be VGA or DVI output from computer
example model: Optoma EP721

Lights
2 small lights (Littlite L-3/12) for mounting on pedestal (5W halogens on 12” goosenecks)
2 very focused lights for ceiling mounting (with lens ideally for sharp focus)

Cables
firewire 400 6-6 cable (15 foot) (camera to computer)
VGA cable to run from computer to projector (50 foot allows flexibility in installation arrangement

Software Installation and Tuning

Software Components

The Giver of Names software includes “The Giver of Names Mac OS X” application, the knowledge file (with a .dump suffix) and a settings file (with a .script suffix) which can be generated from within the program. The .dump file and .script file should reside in a folder called “Lexicon” that should be at the same level in the folder hierarchy as the application. On startup, the Giver of Names loads any .dump files, waits approximately 5 seconds then loads any .script files. The script files are executed to initialize certain operation variables and the settings of the camera. The “GONSettings.script” file generally ends with “GiveNames GiveNames” which starts the normal operation of the software, and hides menus and cursor.
The software relies on the presence of the Mac OS X Carbon Speech Manager (10.5 +) for speaking. It uses open source IIDC camera software to talk over firewire to the camera. An application called “Shadow Killer” is a recommended but not necessary addition. This application removes the 3D shadows that windows cast in Mac OS X. This prevents the projection window form casting a slight shadow on the display window.

The application uses a TrueType font called Humanist supplied in a font file called “Humanst521”. The full name of the font file is Humanist521CondensedBT.ttf. This font file should be in the /Library/Fonts/ folder of the computer’s home folder. In the absence of this font, a similarly condensed font like Arial Narrow or Helvetica condensed could be used.

The minimum system is Mac OS X 10.5. The software is provided as a Universal Binary, allowing it to run on Intel and PowerPC processors. The complete source code is also provided.

There are 2 dump files provided with the software. One is called “GON_x86_32.dump”, and the other is called “GON_x86_64.dump”. The x86 refers to the fact that this was produced for intel processors (little endian) and the 32 and 64 refer to whether they were produced for 32-bit or 64-bit environments. The “GON_x86_32.dump” is the correct one to use for the Mac OS X application as provided.

Minor modifications to the Giver of Names are possible through .lex files. These files are loaded after any .dump file and before any .script file and can contain any command in the GON command syntax (see Appendix C). This will allow for changes to the font used, for example.

Installing the Software

The software is made up of an application (“The Giver of Names Mac OS X”), a folder called “Lexicon” that contains a .dump file containing the knowledge (“GON_x86_32.dump”), a special font (“Humanst521”) and a startup script (“GONSettings.script”).

The Lexicon folder and the application must be in the same folder. That folder would usually be in the home folder of the computer. The font file should go into the /Library/ Fonts folder of the home folder. The startup script is put into the “Lexicon” folder when you want the application to automatically start giving names when it starts.

The “Equipment Details” section below has information on the preferred System Preferences settings.

Once the software is installed, you can simply double click on the “The Giver of Names Mac OS X” application icon to start the program.

Tuning the Software
There are 3 major settings that must be set for the software to operate properly. These are “White Balance”, “Exposure” and “Black Threshold”. These settings are easily adjusted automatically with some simple menu commands. But it is useful to understand what these settings do in order to be able to determine if something is not set properly.

White Balance is an adjustment to the colour sensitivity of the camera so that what we see as white is truly white to the camera. What is considered to be white depends a lot on the lighting in which we see the white thing. Light has what is called a colour temperature. Incandescent light is quite orangy compared to sunlight which is, in comparison, quite blue. Setting the white balance of a camera involves showing the camera something white and asking it to adjust itself so that it sees it as white as well. If the white balance is not set properly, white things will seem bluish or orangish.

Exposure is an adjustment to the light sensitivity of the camera. In can involve adjusting the shutter speed of the camera, the exposure setting of the camera, and the internal gain of the camera. The goal in determining the proper exposure is to keep blacks as black as possible without losing detail in dark areas, and keeping the whitest things as close to pure white as possible without losing some of the shading detail in the white areas.

Black Threshold sets a level which is used by the software to decide whether the pedestal is empty or not. Black is not always pure black. It usually has some noise in it which the software might think is an object. If the camera exposure is too high, black can appear to the camera to be grey or even in extreme cases, white. The Black Threshold sets a minimum amount of brightness in the image to indicate that there is actually something there on the pedestal.

Making the Adjustments
All of these settings are adjusted by the software. When the installation is first fully setup, with the camera and pedestal in place and the lighting adjusted, start the software with the startup script removed from the folder. Selecting “Camera View” from the GON menu will show the camera feed in real-time. Turn on the “Show Pedestal Guides” option in the GON menu to show an outline of the preferred pedestal top

position. Adjust the camera angle and zoom / focus to have the pedestal fill the outlined area at the bottom of the image, and then tighten the tripod screws and lens thumbscrews. Open the lens iris wide. Place a large white object on the pedestal (that fills at least a quarter of the projection) and select the “Exposure Adjust”. The camera will adjust for white balance and will work out the best exposure. To reiterate, this must be done with all the lighting in the room in the final state, and work lights off.

After the Brightness Adjust operation has been performed, clear the top of the pedestal, and select the “Black Threshold” command from the GON menu. This will calculate the threshold for the current lighting environment. If the level is set wrong, the software will take pictures of the black pedestal and try to analyze all the details of the dust and dirt on the felt. This is not desirable. It should not take any pictures if the pedestal is empty. The resulting settings can be saved to be reloaded on startup by using the “Save Settings” menu command in the “GON” menu.

Startup Script
The “GONSettings.script” created by the “Save Settings” command contains commands to set the camera to the desired state, adjust some of the thresholds, and then cause the program to start its normal process of giving names. Other commands can be executed on startup as well, usually by creating a new text file ending in .lex, containing Console style commands that set the desired parameters. For example, if the Humanist font which is usually used by the program were for some reason no longer usable, a .lex file containing the command “SetFont <fontName> <fontScale>” placed in the “Lexicon” folder would set the font to the selected font on startup. The .lex files are always read after any .dump file and before any .script file. You could also modify the .script file, but the next time you use the “Save Settings” command, the new script file would not contain your edits.

System Preferences
Certain system preference settings should be set for proper functioning of the Giver of Names.

System Preferences: Energy Saver :
- Both display and machine sleep set to Never.
- Hard drive may be allowed to sleep (optional)

Under Energy Saver Options:
- Restart automatically after power failure: YES
- Allow power button to sleep computer: NO

System Preferences: Desktop and Screen Saver:
- Screen Saver should be set to Never.

System Preferences: Dock:
- Dock Hiding should be enabled.

System Preferences: Displays

Under Arrangement, mirroring should be disabled.

System Preferences: Accounts:
Login Options:
- automatic log in should be enabled to allow the computer to start without a password.

Login Items:
- the Giver of Names application should be set as a log in item so that the installation starts automatically on turning on the computer.

The Giver of Names does not need a network connection.

Maintenance Protocol

The toys and objects require no special treatment, no condition reports, but as toys get lost or damaged, they should be replaced with other suitable objects. There are certain kinds of objects that are important to have in the and certain qualities that are desirable in an object. These are discussed elsewhere in this manual.

The pedestal may require redraping at some point, and before each exhibition, it is desirable to remove any dust and dirt that may have accumulated in the velvet. The lights attached to the pedestal will eventually burn out and a small stockpile of bulbs is recommended. In future, incandescent bulbs may not longer be available. Current LED lighting is generally too cold to work with the Giver of Names, but should LED lights be used to replace the pedestal lights in the future, the primary consideration is that the colour temperature of these lights should be close to that of the ceiling lights used to light the pedestal.

It is advisable to establish an annual inspection of the core technical equipment. As the computer, camera and projector may become obsolete at some point in the future, and replacements hard to find, it is a good idea to discover any equipment failures as soon as possible, so that replacements may be found or repairs made while this is still possible. It is also recommended that an annual survey be made of the status of the computer, its operating system, the camera type, and the type of connection made to the projector. The aim would be to keep an eye on when the technologies used in the Giver of Names approach obsolescence, so that backups, where required and desired, can be obtained easily and cheaply. This sort of research could easily and should best be performed by someone with some technical expertise.

Things to check:
Status of Mac OS X. Generally, new operating systems provide some level of support for running software designed for earlier systems. The support ranges from complete and comprehensive to spotty. If a substantially different operating system takes the place of OS X, then it would be important to explore whether the Giver of Names software will run successfully on the new OS.

Status of Firewire (aka iLink) (aka 1394) This is the kind of connection between the IIDC camera and the computer. Future computers may support the software, but not provide a firewire port, in which case, an alternative camera type would be required.

Status of VGA for display connections. Future projectors and/or computers may use digital only connections. This may mean that a change in computer might require a change in projector, or the purchase of some sort of adapter.

Source Code Summary

The source code is provided so that, if necessary, the software can be adapted to other operating systems. A programmer with general facility in programming applications should be able to port the source code to a new operating system fairly easily.

The code is divided into 3 sections: Core code, abstract implementation glue and Mac OS X 10.5 Cocoa implementation files.

The GON Core code makes minimal operating environment assumptions. It uses Unix calls for file system actions, and standard C and C++ calls for everything else. It is written mostly in C++ with some routines in C. The Core code can be compiled into generic command line tool that provides most of the core functions of the Giver of Names.

The implementation glue is made up of a small number of C++ classes that provide and empty interface between the core and the specific implementation classes that define the Operating System dependent parts of the application. These classes define an API.

The actual implementation files include subclasses of all the glue classes with actual code in the member functions. In the case of the Mac OS X Cocoa implementation, these classes also handle a transition from C++ to Objective-C. The rest of the implementation classes are probably unique to any particular target OS... they control the GON core entirely through the glue classes and in return, respond to calls form the core.

The implementation glue classes cover 4 main functions: Application wrapper, graphics display, image capture and speech synthesis.

The application wrapper is a bare-bones application which is responsible for instantiating the core, and offering menu control over certain functions. The graphics display classes communicate from the Core to whatever graphic system us used on the host OS.

In the Mac OS X implementation, the graphic display functions are ultimately executed by the Open GL graphics library of the OS. Window and View management is largely handled by generic Cocoa classes like NSWindow and NSView, or minor subclasses thereof. Image capture is handled with Open Source IIDC libdc1394 code that is at the

time of writing extensively cross-platform. Speech synthesis is effected by the Mac OS X built-in Carbon Speech Manager. The source code was compiled as a project using Apple's XCode 3.1 development environment.

In all cases, the glue has been made as minimal and generic as is reasonable, leaving lots of scope for building the implementation code according to one's tastes and preferences.

A much more extensive reference guide to the source code is presented in the **Giver of Names Software Technical Introduction** and in **Appendix B.**

The Giver of Names Software Technical Introduction

Overview

The software is organized into 3 basic sections: Core, Abstract Implementation, and OS Specific implementation.

Core performs all the basic functions of the Giver of Names and is written in standard C ++ and avoids any assumptions about the operating system that it is running on. Core can be easily wrapped into a console application, allowing the main language and database functions to be ported to other systems with minimal effort. A command line version of core has been tested in a 64-bit context, so there should be no issues with 64-bit incompatibilities. Endian issues are essentially limited to the .dump files that contain a compressed version of the Knowledge base and are not transparent to endian-ness and 32-bit vs 64-bit operation. Dump files can be easily constructed for a new situation. (See **Appendix c: The Giver of Names Command Syntax** for details on the 'Dump' command)

An abstract set of implementation classes are provided to create a generic glue between operating system-specific functions and Core.

An operating system specific implementation communicates back and forth with Core via the abstract implementation.

It is likely that any reimplementation of the Giver of Names would not involve any modifications to the Core. The following descriptions and details may be useful, but are not necessary for normal porting.

Core

The Giver of Names core software includes the following main components:

cKnowledge
cArticulator
cPerceiver
cAssociater

cKnowledge

cKnowledge holds the knowledge base for the Giver of Names, and manages the access, navigation and stimulation of it. The knowledge base is full of words, ideas, grammar trees, property definitions, as well as commands for manipulating the data. The data is originally defined in text files which are parsed by cKnowledge in order to construct the knowledge base. The parsing of the full knowledge base can take several hours. The database can also be saved back out to disk as a single text file. A compressed version (knowledge dump) of the knowledge base can be dumped to hard-drive and read back in on application launch in a matter of seconds.

cArticulator

The cArticulator class manages the language generation for the Giver of Names. It depends on syntactic and semantic information in cKnowledge. It builds sentences using grammar rules held in cKnowledge and fills the various slots in the sentence structure from words held in cKnowledge, often chosen because of their level of stimulation.

cPerceiver

The cPerceiver class manages the perception side of the Giver of Names. It controls the video source, does all the image processing on the source images, and parses the segments of the image into a cPictureTree in conjunction with cPictureParser. This tree is used by cArticulator as an aid in structuring sentences and establishes relations between components of the image that may be reflected in the sentence contents.

cAssociater

The cAssociater class was intended to coordinate communication between the cKnowledge, cArticulator and cPerceiver. In practice, these objects are often communicating directly with each other. cAssociater does control the initial reading of data and settings files, but mostly stays out of the way during actual processing.

Greater detail on these Core classes is provided in Appendix B.

Porting Process

The process of porting the Giver of Names would probably progress as follows:

1/ compile and test command line version of Core. You will probably have to load the full .lex file rather than a .dump file if there are significant differences in architecture (i.e .endianness, 32-bit vs 64-bit). In this case, you also want to make a .dump file of the knowledge base for the new architecture using the Dump command from the console so that you do not need to re-parse the .lex file while testing each new component. (See **Appendix c: The Giver of Names Command Syntax** for details on the 'Dump' command)

Note that the prefix file (.pch) is set up to allow you to strip down to the abstract implementation for initial testing and add implementations of each set of classes as a group for incremental porting.

2/ implement and test a subclass of GONAbstractApplicationGlue for the target operating system.

3/ implement and test a subclass of GONAbstractConsoleWindow and any necessary additional classes to get a functional console window functioning in an application wrapper native to the destination operating system.

4/ implement and test a subclass of GONAbstractProjectionWindow and any necessary additional classes to add the projection window to the application.

5/ implement and test a subclass of GONAbstractDisplayWindow and accompanying classes.

6/ implement and test voice and image capture solutions for the target operating system using GONAbstractVoice and GONAbstractVideoSource.

Abstract Implementation

The abstract implementation provides a set of largely non-functional base classes that present the necessary glue interface to the Giver of Names Core from the perspective of a host application. Wherever possible, generic functions that are independent of operating system are placed in the abstract implementation classes, so that only the most generic application level programming must be redone for a new implementation.

Abstract files and classes

GON_OS_Glue

This is a header file detailing some basic operating system functions that the Giver of Names Core requires. These functions need to be implemented for the native operating system.

For the initial Cocoa implementation, these functions are implemented in Unix_Glue.cp and Mac_OS_X_glue.cpp.

FUNCTIONS TO BE CUSTOMIZED IN THE OS IMPLEMENTATION

```
char *AllocPtr( size_t size )
```
Allocate a chunk of memory of specified size and return a pointer to it. Return Null if fails.

```
char *AllocPtrClear( size_t size )
```
Allocate a chunk of memory of specified size, clear it to all zeros and return a pointer to it. Return Null if fails.

```
void DeallocPtr( char *pointer )
```
Deallocate a chunk of memory.

```
int16_t GetApplicationBundlePath( char *pathName, int32_t maxLength )
```
fills pathName with the full path to the directory containing The Giver of Names application (maxLength indicated the maximum number of characters that pathName can receive.)

```
int16_t LocateFolderInGONPath( char *folderName, char *folderPath, int32_t
maxCount )
```
Locates a specific folder (directory) inside the folder containing the Giver of Names Application and sets it up for iterating through. This is used to iterate through the “Lexicon files” folder on startup.

```
bool GetNextFileInFolder( char *fileName )
```
After a “LocateFolderInGONPath” call is made, this function is called repeatedly to get the names of each file in folder. It returns false if there are no more files to be found.

```
int16_t SwapToBig16( int16_t input )
int32_t SwapToBig32( int32_t input )
```
These handles endian-ness. They takes the input and swaps it from the operating system's native endianness to big endian. NOTE: These are only used in one function in cImageFile to parse the header data for an image read from disk. This is only useful for testing and does not need to function correctly for correct function of the Giver of Names.

```
void Wait( float delay )
```
This waits in a loop until delay seconds have elapsed.

```
void WaitTicks( float ticks )
```
This waits in a loop until 'ticks' 1/60ths of a second have elapsed.

```
double CurrentTimeInSeconds( void )
```
This returns the current time in seconds. It does not matter what the reference is. This is only used for relative timings, so it could be relative to the time the computer booted up, or relative to some specific time and date in the past, for example.

GONAbstractApplicationGlue

This class is an abstract that provides a glue for basic application operations. For the initial Cocoa implementation, this is subclassed by CocoaGONApplicationGlue.

The source file also creates global objects for the 3 windows: gConsoleWindow for communication back and forth with the core, gDisplayWindow for the display to the computer screen, and gProjectionWindow for the display to the projection. These global pointers are used from within the Core to call for things to be displayed.

The following virtual routines will need to be overridden:

```
virtual void GONUpdateSpeechDisplays( void )
```
This causes the display and projection windows to update their contents if needed.

```
virtual void StartGivingNames( void );
```
This starts the process of capturing images, analyzing them and describing them in a continuous loop. This glue routine is called by cKnowledge when it encounters a "GiveNames" command in a startup script.

```
virtual void StopGivingNames( void );
```
This stops the process of capturing images, analyzing them and describing them in a continuous loop. This is not generally used. The Application generally has a menu option that does this same thing.

```
virtual void SetFont( char *font, float scale )
```
This function sets the font name and relative scale of the fonts used in the Display and Projection window. The standard font is Humanist521BT-RomanCondensed and the standard scale is 1.0. This font is smaller than most in relative size, so if a replacement font is used, its scale will likely in the order of 0.6 to 0.75. The font is intended to be narrow or condensed, so that more words fit accross a line. ArialNarrow is a possible replacement (at a scale of about 0.7)

IMPLEMENTED FULLY IN THE ABSTRACT CLASS:

```
virtual void InitApplication( void )
```
This creates the windows, then creates a cAssociater object that open up any datafiles and scripts that are in its “Lexicon Files” folder.

```
virtual void QuitApplication( void )
```
This deletes the display and projection windows and the cAssociater object, which in turn deletes the cKnowledge, cArticulator, and cPerceiver objects.

```
virtual void SaveSettings( void )
```
This sends a command to cKnowledge which results in saving key variables such as camera settings into a script file called “GONSettings.script” which is saved in the “Lexicon Files” folder and will be executed each time the Giver of Names application is started up. This settings file can be edited by hand to set other parameters.

```
virtual void GONIdle( void )
```
This calls the Giver of Names Core Idle function. This function is mostly used at startup to manage a timed delay between application startup and the running of a startup script.

GONAbstractDisplayWindow

This class provides an abstract base class for communications from the Giver of Names core to the display window (the one that occupies the computer screen).

There is considerable communication back and forth between the Giver of Names Core code and the display window. The Core contains some specialized drawing routines to implement the specific form of compositing that is used for this window. The display window provides operating system dependent services like drawing text into a buffer, updating the images to the screen, etc.

The GONAbstractDisplayWindow and any subclasses that override it for a particular implementation do not necessarily perform all the tasks themselves. In the Mac OS X implementation, GONAbstractDisplayWindow and its sub-class CocoaDisplayWindow merely function as a portal through which the Core communicates with display system for that window. Several other classes are used as suits the operating system for the

windows themselves and any “views” and offscreen buffers that may be required to perform the desired tasks.

The display window has 2 main images displayed on it. A stimulation image is used as the background of the window, filling its frame. This image is build up of many words, drawn in various colours, overlaid to produce a sort of text cloud. The buffer for this image is created and owned by the display window or some other class in the implementation. The Giver of Names Core asks the implementation via this glue object to render a word in a specific colour into a small buffer (also managed from the implementation). The Core then asks for the buffer, wrapped as a cImage. The Core then manages superimposing this new text over the old by comparing each pixel of the new text and the accumulated stimulation image, writing the brightest of the two back to the stimulation image. Then the Core asks the display window to update the stimulation display.

Over this background, the recent spoken text is displayed, with the most recent text at the bottom and previous texts scrolling up and out the top of the window. This text is received in chunks from the Giver of Names Core as it is spoken.

For the initial Cocoa implementation, this is subclassed by CocoaDisplayWindow. See the desciption of CocoaDisplayWindow and its related classes to see how this is implemented in the Mac OS X implementation.

```
virtual void UpdateGONDisplayView( void )
```
Update the display window now.

```
virtual void DrawStringImage( cImage *stringImage, char *name, pixel32
colour )
```
Draw an image of the supplied string and return the image in the supplied cImage structure, in the colour defined by the colour parameter. The cImage structure passed is effectively empty. The DisplayWindow must allocate this buffer and fill in all the other cImage members to suit.

```
virtual bool DisplayExists( void )
```
Returns true if the display window does actually exist. This is tested before attempting to draw to the window.

```
virtual void CaptureStimImage( cImage *stimImage )
```
This fills in the supplied cImage object with the full screen stim buffer and relevant details from the offscreen image of the stimulated words that is the background for the display window.

```
virtual void SetStimImageChanged( void )
```
Sets a flag to indicate that the stimulation image has been changed and should be updated.

```
virtual void FrameSelectedStim( cImage *stimImage, cImage *stringImage, float
h, float v )
```
Draws a frame around a particular word in the stimulaltion image.

```
virtual float GetStimFontSize( void )
```
Returns the appropriate size for the stimulation font given the current screen size.

```
virtual void UpdateFont( void )
```
Tells the display window that the font has been changed and so the display parameters should be changed to suit.

```
virtual void GetDisplayWindowDimensions( float *width, float *height )
```
Returns the current display window dimensions.

```
virtual void ClearStimImage( void )
```
Clears the off-screen buffer that contains the stimulation image.

```
virtual void FreezeStimDisplay( void )
```
Signals that the Stimulation image should not be updated currently even if the contents change. This is because often many words are drawn in rapid succession and it would be extremely inefficient to update the screen for each instance.

```
virtual void UpdateStimDisplay( void )
```
This unfreezes the stimulation display and causes an update of the stimulation image in the window.

```
virtual void FillPrimaryScreen( void )
```
This causes the display window to fill the primary computer screen leaving no borders or window title.

```
virtual void MoveToFront( void )
```
This should move the display window to the front of the window layers.

GONAbstractProjectionWindow

This class provides an abstract base class for communications from the Giver of Names core to the projection window.

The projection window is primarily used to display the camera image and the analysis of that image as it progresses. It also shows the text as it is spoken across the bottom of the image with the currently speaking section of the test being highlighted and displayed centered in the window.

For the initial Cocoa implementation, this is subclassed by CocoaProjectionWindow.

As with CocoaDisplayWindow a number of classes are used to actually implement the functions that are effected through this glue interface, the nature of these will depend largely on the preferences of the programmer and the nature of the api of the operating system being ported to.

```
virtual void ShowSentenceToWord( char *sentence, int16_t startOfChunk,
int16_t sizeOfChunk )
```
This function displays the section of text from startOfChunk for sizeOfChunk characters in the center of the image, in white, with the section of the sentence up to startOfChunk running off to the left with a diminished alpha.

```
virtual void FinishSentence( void )
```
This sets up a fade of the alpha on the currently displayed end of sentence so that it fades over time to invisible. It should fade out in about 3 seconds.

```
virtual void ShowImageInProjection( cImage *inImage )
```
This function displays the supplied cImage (buffer provided by the Core) in the full projection window as the background.

```
virtual void DrawOutlineInProjection( cPointVector *inOutline, float inRed,
float inGreen, float inBlue, float inAlpha, float inSize )
```
This function draws an outline over the image, outlining the particular object in the image, scaled to match the scale of the image. cPointVector is a structure containing the vertices of the polygon to draw. inSize defines the width of the line to draw (which should be wide enough to clearly highlight the object.

```
virtual void UndrawOutlineInProjection( void )
```
This causes the current outline to cease being drawn.

```
virtual void ShowPedestalGuides( bool is )
```
This displays guides in the projection window that indicate the profile that the pedestal should present in the image coming from the camera. It is used only during setup. The pedestal left edge should be .16666666/2.0 x the width of the window and the right edge should be the same distance from the right side of the window. The top guide should be at 1/6 of the height of the window above the bottom of the window.

```
virtual void FillSecondaryScreen( void )
```
This causes the projection window to fill the secondary screen (the one shown on the projection.

```
virtual void MoveToFront( void )
```
This moves this window to the front of the window layers to be visible.

GONAbstractConsoleWindow

This class handles communication between the console window and the Giver of Names Core. Commands typed into the console window are parsed and executed by the Core code. Results of those commands, and any other console messages from Core are displayed in the console window. Commands are executed in the following ways: Selecting a chunk of text and pressing the enter key dispatches the selected text to the Core. Typing enter at the end of a line of text dispatched the line to the Core. Placing the cursor at the end of a line of text dispatches the line to the Core. Output is conventionally placed at the bottom of the console text, although the commands can be typed anywhere in the console window.
This abstract class is subclasses by CocoaConsoleWindow in the Mac OS X implementation.

THESE FUNCTIONS WILL NEED TO BE OVERRIDDEN

```
virtual void AppendTextToConsole( char *text, int32_t length )
```
Sends the text to be displayed at the bottom of the console window.

```
virtual void MoveToFront( void )
```
This should the console window to the front of the window layers.

IMPLEMENTED FULLY IN THE ABSTRACT CLASS:

```
void SendCommandForProcessing( char *text )
```
Passes the received zero terminated string to the cAssociater object to be parsed and executed as a command.

GONAbstractVideoSource

This is a true base class for the video source, using virtual functions to implement os specific code. The video source must be able to initialize itself to an appropriate setting for capture and return video frame on request.

For the initial Cocoa implementation, this is subclassed by IIDCVideoSource.cpp.

FUNCTIONS TO BE OVERRIDDEN IN THE OS IMPLEMENTATION

```
GONAbstractVideoSource( cPerceiver *inPerceiver );
```
construct a new video source.

```
GONAbstractVideoSource( void );
```
delete a new video source.

```
virtual bool GetFrame( cImage *outImage );
```

start capture if not running, request a frame, wait for it, convert it to a 640 x 480 pixel 8-bit per component ARGB image and then package it as a cImage. (the internal image structure for cPerceiver)

```
virtual bool CaptureReady( void );
```
returns true if the capture device is ready to capture

```
virtual void EmptyBufferQueue( void );
```
clears out any frames in the buffer queue of the capture device.

```
virtual void RequestNewFrame( void );
```
requests a new frame from the device

```
virtual bool GetNextFrame( cImage *outImage );
```
Assuming that a new frame has already been requested, get the next available frame, convert it to a 640 x 480 pixel 8-bit per component ARGB image and then package it as a cImage. return false if no frame was available.

```
virtual void SetWhiteBalance( void );
```
where the video source supports it, causes the camera to do a white balance.

```
virtual void AdjustExposure( void );
```
where the video source supports it, causes the camera to set its automatic parameters to manual and to select an optimal shutter speed.

```
virtual void SetParameter( char *codeName, char *modeCodeName, float *values,
int32_t valueCount );
```
where the video source supports it, set a named parameters to be in a certain mode and with the specified values.

```
virtual void ReportFeatures( ostream *out );
```
where the video source supports it, report the controllable parameters of the camera or capture device.

```
virtual void ReportCameras( ostream *out );
```
where the video source supports it, report the cameras that are connected.

```
virtual void PrintSettingsCommand( ostream *out );
```
print the current settings to the provided stream. This is intended to print the parameters in a format that can be parsed by the command parser of cKnowledge. If the stream is output to a file, reading and parsing this file would set the camera to the settings as that were printed.

This is currently implemented as a command with the syntax:

iidc <parameter name> <optional mode (manual/auto/one_push/absolute)> <value> <optional additional value> <optional additional value>

For example, the correct format for a manual shutter setting of 30 would be

```
iidc shutter manual 30
```

and the correct format for adjusting white balance then freezing it would be

```
iidc white_balance one_push
```

The command does not assume that the camera is actually an iidc camera. The parameter name, mode and values are passed along to your SetParameter function as is. You can do with them what you would like.

```
virtual void PrintParameter( char *codeName, ostream *out );
```
print the current setting for the specified parameter to the specified stream. (uses the same output format as PrintSettingsCommand.

```
virtual void ResetCamera( void );
```
resets the camera to a default state.

IMPLEMENTED FULLY IN THE ABSTRACT CLASS:

```
bool GetBlendedFrame( cImage *outImage, int32_t frameCount, float
washoutFraction, float maxGain, float noiseThreshold, float
motionThreshold );

bool GetBlendedFrameNoTests( cImage *outImage, int32_t frameCount, float
washoutFraction, float maxGain );

float AccumulateFrame( cImage *inSource, float threshold );

void ScaleAccumToImage( cImage *dest, float maxGain, float washOutFaction );
```

These functions are implemented fully in the abstract class and call the GetNextFrame function of the operating system specific implementation.

GONAbstractVoice

GONAbstractVoice provides a base class for communication with a voice synthesizer.

For the initial Cocoa implementation, this is subclassed by cCarbonSpeech.cpp.

FUNCTIONS TO BE OVERRIDDEN IN THE OS IMPLEMENTATION

```
void SetVoice( char *voiceName )
```
selects the named voice

```
void Speak( char *text, int32_t length )
```
speak the text in the buffer up to the specified length.

```
void ShutUp( void );
```
stop speaking

```
bool Busy( void );
```
returns true if the voice is still speaking the previously supplied text.

```
int16_t SetRate( double theValue );
```
sets the rate of speech (approximately in words per minute)

```
int16_t SetPitch( double theValue );
```
sets the base pitch for speech, with 42 sent for male voices and 52 sent for female voices.

```
int16_t SetModulation( double theValue );
```
sets the voice modulation degree. Not actually called.

```
int16_t SetVolume( double theValue );
```
set the volume of the speech. (1.0 is normal gain)

```
int32_t InstallWordCallback( void )
```
This function sets up a callback of some sort from the speech engine to indicate what part of the text is currently being spoken. The callback must set the values of `currentSpeechPos` and `currentWordLen` referring to the start position of the word currently being spoken and the length of the word in characters. The Giver of Names software uses these values to display the sentence as it is being read. Note that the Giver of Names code sets currentSpeechPos to -1 after displaying some text so that it can tell when the next updated position is ready.

Get functions for rate, pitch, volume and modulation are in the api but do not need to be implemented.

Mac OS X Implementation

Development Environment

The Mac OS X implementation is based on Cocoa, the Objective-C framework that is standard on the Mac OS. The implementation was compiled using Mac OS X 10.5 headers using XCode 3.1.1. using the GCC (Gnu CC) version 4 compilers. XCode allows Objective-C and C++ to be mixed. Mixed source files have .mm as their suffix. Objective-C source files normally have .m as their suffix. The Abstract Implementation code is entirely written in c++. The use of Objective-C is specific to the Cocoa implementation. It is assumed that anyone porting the Giver of Names would use their preferred language and frameworks for the target OS, ignoring Objective-C except when looking at the Cocoa implementation for guidance.

Micro intro to Objective-C

The following syntactical peculiarities of Objective-C are useful to know when examining the Cocoa code for examples of how things might be implemented.

1/ Objective-C does not use the C/C++ convention of calling member functions in this manner:

```
object -> function( arg1, arg2 );
```

It does the same thing with the following syntax:

```
[object function:arg1 secondArgumentName:arg2];
```

In Objective-C, each argument has a name explicitly stated in the function call.

2/ The syntax of function declarations is not

```
returntype function( arg1type arg1name, arg2type arg2name )
```

but

```
- (returntype) function: (arg1type)arg1name (arg2type)arg2name
```

3/ Objective-C uses generally uses #import instead of #include (to do the same thing)

4/ Objective-C uses surrounds the class definition in the header file with @interface at the start and @end at the end after the class function declarations, and surrounds the actual function definition section of the source file with @implementation and @end.

Micro Intro To Cocoa

Cocoa is an application framework that comes standard with Mac OS X. It is written in Objective-C. The one challenge to understanding the function of a Cocoa application for someone who is not familiar with Cocoa is that there is often no explicit instantiation of main classes visible in the code. The windows are not necessarily ever explicitly created. This is because Cocoa applications rely on a secondary file called a nib file. Nib files contain information about the key objects that make up the application such as windows, menus, views within windows, etc. When an application launches, it interprets its nib file and creates the objects described there. The nib file also defines some connections between objects, and what messages are sent and in which cases. These messages effectively call functions in the receiving object. So a fair amount of the function of the application is held in the nib file and will not be found in the code.

The following things are constructed by the nib file:
- an application object (NSApplication class)
- a main menu bar and its menus and menu items
- the console window
- the display window
- the projection window
- an offscreen window (the purpose of this window will be explained later)
- a font managing object

In addition, the nib creates some things called controllers, that partner with some of these objects to add functionality. This is a mechanism used by Cocoa as an alternative to subclassing an object. Pairing a controller object with an application or window or view allows you to add behaviour to that object without explicitly subclassing it. It is beyond the scope of this introduction to examine why this is done.

The following controller objects are also created by the nib file:
- a controller to enhance the application object (CocoaController)
- a controller to enhance the console's text display view
- a controller to enhance the text view of the offscreen window

In the description of the Cocoa classes, I will describe the connections that may be hidden in the nib file.

Mac OS X Implementation Classes

Unix_Glue
This implements some of the GON_OS_Glue functions using standard Unix calls.

Mac_OS_X_glue
This implements the rest of the GON_OS_Glue functions using Mac OS X specific calls.

CocoaGONApplicationGlue
This provides simple glue routines to the Cocoa Application, and overrides the GONAbstractApplicationGlue class. This mixes Objective-C and c++, and therefore has an .m suffix. Its only function is to provide a C++ interface to the Cocoa-based CocoaControllerClass.

CocoaController
This is a controller object that manages the core functions of the actual appplication around the Giver of Names core. Its primary jobs are to initialize the application, respond to menu commands, and to schedule the Core's process of grabbing images and responding.
Menus

The Edit, Format, and Window menus are standard Cocoa menus and are not needed for the appllication, although the Window menu is useful for revealing hidden windows. The Window menu conventionally provides a list of the open windows and allows you to bring one to the front. Because the display and projection windows have been set up to be full screen and have not title bar, they are not included in the Window menu. In Mac OS X, you can cycle through an application's windows by pressing (cmd-`) repeatedly, which will eventually reveal the desired window.

The GON menu is the only important menu in the menu bar, and the functions contained there could be implemented in some other way than menus theoretically.

Kiosk Mode
This command toggles kiosk mode on and off. When Kiosk mode is on, the menu bar hides itself unless you move the mouse pointer to the top of the screen. Kiosk mode is automatically entered when the Giver of Names starts processing.

Run
This tells the Giver of Names to start doing its thing: repeatedly grabbing images, analyzing them and talking about them. When using the menu item to stop running, there may be a delay while the Giver of Names finishes its cycle.

Save Settings
This tells the Core to save key settings to a settings files that can be read on startup to set the proper parameter settings for that camera and various other operational settings

Grab Image
This grabs an image from the camera (actually the average of a series of 8 images)

Grab and Name
This grabs an image, then analyzes it and constructs and speaks a sentence, then stops.

Camera View
This shows the live output from the camera

Show Pedestal Guides
This draws guides on the live camera view that help in positioning the camera and setting the appropriate zoom on the lens.

Determine Black Threshold
This is used to determine what the empty pedestal looks like so that the Giver of Names does not try to analyze the random noise in the darkness.

Adjust White Balance
This tells the camera (where the camera has this function) to take a white balance and then freeze the white balance setting.

Adjust Exposure
This does a white balance, sets the camera to manual mode, and adjusts the shutter speed to its optimal setting, then does a final white balance.

Camera Settings
This is just a place-holder but might be useful if some other form of video source is used.

CocoaConsoleWindow
This is a subclass of GONAbstractConsoleWindow. It does not create or own the actual console window... it just provides a glue interface to the actual Cocoa window.

Its only function in this implementation is to override the AppendTextToConsole function to actually append the text.

The actual console window is a standard Cocoa NSWindow filled with a CocoaTextView (a subclass of standard Cocoa NSTextView) contained inside a scrolling view with scrollbars (a standard Cocoa NSScrollView). This combination of objects simply creates a window containing text that can scroll.

CocoaTextView
This class inserts any text received via CocoaConsoleWindow**::**AppendTextToConsole into the window's text view. On creation, it also makes a connection with its controller, CocoaTextViewController.

CocoaTextViewController
Most of this class is basic Cocoa specific glue. The only important function in this object is the one called doCommandBySelector. The job of this function is to handle situations where the user presses the enter key or return key in the console window. The controller automatically receives notification of these events through this "doCommandBySelector" function. The function gathers the current selection in the console window and passes it to the GONAbstractConsoleWindow object to pass to Core for processing as a command. If the selection length is zero, it gathers the entire line containing the selection point.

CocoaDisplayWindow
This is a subclass of GONAbstractDisplayWindow. It does not create or own the actual display window. It provides a glue interface for that window.

The display window shows 2 things: a stimulation image as the background and the scrolling field of recently spoken text in front.

This implementation creates two offscreen images, turns them into OpenGL textures and then composites them into the window. OpenGL is used here because it was the simplest way in this context. The scrolling field of recently spoken text is rendered into an offscreen window, then the image of the window is captured. It happens that, in this case, the alpha channel of the image data is zero for the backgound and 1 for the text, so that this gives the necessary transparency for the stimulation image to show through the spaces between the letters.

The actual class used for the stimulation image is:

QuartzOffscreenStimImage
This uses the Quartz graphics system standard to Mac OS X. This object creates two offscreen image buffers. One small one is used to render single words. The second, larger buffer holds the entire stimulation image. This class has a tight relationship with Core, as Core manages the actual compositing of words into the stimulation image. Core will ask QuartzOffscreenStimImage to render a word, then ask for an up-to-date image of the stimulation image, and then does the compositing itself. Then it also asks QuartzOffscreenStimImage to frame the words when selecting them for a sentence. Core will also ask QuartzOffscreenStimImage to clear the stimulation image when required. QuartzOffscreenStimImage provides its internal buffers in cImage structures that Core can use.

The classes used to implement the spoken text display are:

CocoaOffscreenSpokenTextViewForDisplay
This is a subclass of Cocoa's standard NSTextView. It received spoken text from Core via the AppendTextToDisplayOverlay function of GONAbstractDisplayWindow (actually its subclass CocoaDisplayWindow). It inserts the text at the insertion point, which will always be at the end of the text. It scrolls the text so that the insertion point (the end of the text) is in the visible frame of the NSTextView (though it is not at this point visible on screen because this view is in an offscreen window). The image of this view is converted to a texture for actual display by CocoaTextTextureForDisplay. It is responsible for setting the font and font size for the display. The font used for the Giver of Names is a TrueType font called Humanist (the internal name required in the implementation is "Humanist521BT-RomanCondensed". If this font is not available it can be replaced with a condensed (narrow) san-serif font with medium weight.

CocoaSpokenTextTextureForDisplay
This class is responsible for creating an OpenGL texture, capturing the image of the CocoaOffscreenSpokenTextViewForDisplay into the texture, and drawing the texture. It is informed when the CocoaOffscreenSpokenTextViewForDisplay contents change so that it can update the texture if necessary before drawing.
The texture is is displayed in the display window 20 pixels down from the top and 20 pixels in from the left.

The view into which all display window stuff is drawn is:

CocoaDisplayGLView
This class is the view in which the stimulation image and spoken text are finally drawn, It fills the display window. It sets up the OpenGL display environment, handles resizing that environment if the window changes size, and draws the textures by calling the draw functions for both QuartzOffscreenStimImage and CocoaTextTextureForDisplay. If stimulation image is frozen during updating, the view is not drawn even when there has been some change in the stimulation image. The draw routine is called from a Mac OS X DisplayLink. This causes the redraw to happen during the screen retrace.

CocoaProjectionWindow
This is a subclass of GONAbstractProjectionWindow. It does not create or own the actual projection window. It acts as interface glue between Core and the actual projection window objects.

The projection window shows the processed image form the camera. Overlaid on this is the text that is currently being spoken, with the currently spoken word in the center in white, and the preceding sentence running off the left edge, with a diminished alpha value, so that it is less emphasized.

The text is formatted by CocoaProjectionWindow into an NSMutableAttributedString, which is a Cocoa string class that allows attributes like colour to be attached to individual characters, so the highlighting of the currently spoken word is achieved in this way. It can otherwise be done by separating the strings into the end section to be highlighted and the rest, and displaying them in different colours or with different alphas.

The projection window also manages a smooth fadeout of the sentence after it ends by ramping the alpha of the text down to zero.

It also can display the live video source for setup, and show guides that aid in placing the pedestal and adjusting the camera.

The class used for the camera image is:

CocoaProcessImage
This class receives a cImage from Core and copies it into an internal buffer. This image is destined to be a texture for display using OpenGL. This image from Core is 320 x 240 ARGB format. When the CocoaProcessImage is requested to draw itself, it first checks to see if the texture is up-to-date. If not, it generates the texture using the internal buffer. Then it draws itself.

CocoaStringTexture
This class draws the currently spoken text across the bottom of the projection. The incoming string in this case is an "attributed string", meaning that it has colour, font, alpha already defined. This object simply renders the string to a texture and draws itself when called upon to. It also manages an alpha fade for fading out the text after the sentence is over.

CocoaProjectionGLView
This class is the view in which the processed image and string texture are finally drawn. It fills the projection window. It sets up the OpenGL display environment, handles resizing that environment if the window changes size, and draws the textures by calling the draw functions for both CocoaProcessImage and CocoaStringTexture. It also draws any GONOutline objects that are currently to be displayed. The draw routine is called from a Mac OS X DisplayLink. This causes the redraw to happen during the screen retrace.

IIDCVideoSource

The video source in the provided implementation of the Giver of Names uses an IIDC camera. (IIDC cameras provide uncompressed video through firewire (aka iLink, aka 1394) using the IIDC protocol) The Giver of Names code uses an open source iidc library called libdc1394. The source code is included with the rest of the GON code. The version used is libdc1394-2.0.2. IIDC provides some advantages over other options at the time of writing because it allows direct control of the internal camera parameters which make adjusting white balance and exposure much simpler. Any kind of video source could be used. It needs to be able to provide images at a resolution of 320 x 240 pixels in cImage format. cImage format is an internal format for GON that is simply a wrapper around pixel data providing information about the format of the image in memory. cPerceiver controls the video source.

The parameters generally available for IIDC cameras are:

exposure <manual/auto> <value>
gain <manual/auto> <value>
shutter <manual/auto> <value>
white_balance <manual/auto/one_push> <value> <value>
brightness <manual> <value>
gamma <manual> <value>
saturation <manual> <value>
hue <manual> <value>
sharpness <manual> <value>

The preferred setting for the Giver of Names is manual exposure, gain, brightness, gamma and saturation at default levels with the white_balance set by one_push mode while a white object is on the pedestal, and with shutter in manual with a setting determined by the adjust exposure command. The camera has a manual iris which normally should be wide open. The adjust shutter command attempts to find a shutter setting that allows a minimal amount of clipping of the brightest whites.

cCarbonVoice

cCarbonVoice is a speech synthesis implementation using the Carbon Speech Manager for Max OS X 10.5.

This implementation is very straightforward. You can see an example of a word callback routine there.

The Giver of Names Knowledge Base

The Giver of Names knowledge base exists in 3 forms.

First there are a set of lexicon files (ending in .lex) which define the various parts of the knowledge base in editable text files using a standard syntax. These files define the core knowledge base but are missing the T_XXXX properties that were derived form reading various novels.

Secondly, there is a single text file ending in .lex containing a complete knowledge base including the T_XXXX properties.

Thirdly, there is a file containing the entire database in a compressed non-editable form (the name ends in .dump). It is also a form that may have some operating system dependancies in it. For example, endian-ness will matter. The PowerPC version of this dump file was not compatible with the intel version because they place multibyte data into memory in the opposite order. It also make some assumptions about the size of pointers (assuming 32-bit), which may be wrong on a 64-bit system.

The .dump file is useful because it is many, many times faster to load. The .lex files may take hours to load, and so are unsuitable for the startup of the installation in a gallery.

In a new implementation, the .lex file or files could be loaded, then a .dump file can be created for that particular implementation. Look in the Giver of Names command summary for the Dump command description.

On startup, the Giver of Names locates a folder called “Lexicon” and processes the files there in the following order. If there are any files ending in .dump, they are processed first. Then any files ending in .lex are processed. The .dump file sets up a complete cKnowledge structure and any additional .lex files will add additional data to the knowledge base.

Knowledge Base Syntax

The .lex files use a subset of the Giver of Names command language. In fact, only one command is used. The “Declare” command creates a new entry in the knowledge base and adds some properties to that entry. The syntax of a Declare command is as follows:

Declare EntryList PropertySet

The Declare command takes a list of entries (EntryList), and a list of properties (PropertySet), creates an entry for each entry in the list if it does not already exist, and then adds the properties in the list to all the entries. For example:

Declare n house{ Sense _n housing.house; }

This creates an entry of noun category with the name house if it does not exist, and adds a Sense property pointing to the “noun meaning” entry for housing.house (also creating the housing.house entry if it does not exist.

PropertySets are always surrounded by curly braces {} and each property declaration ends in a semi-colon.

Weights can be attached both to the entries in the EntryList and to the properties. For Example

Declare n house@445{ Sense _n housing.house@146; }

The weights can have various significances. Most words have weights related to their relative frequency of use in spoken and written English. Sense properties are weighted in a similar way. The weights sometimes reflect the relative strength of a property. For example

```
Declare n house
{
Kind n mansion@1;
Kind n condominium@10;
Kind n family_home@10;
Kind n apartment@20;
}
```

In this case, when choosing a kind of house using the weights of the properties to shape the probabilities, apartment would be more likely to be chosen than mansion.

The above declaration could be shortened to

```
Declare n house
{
Kind n mansion@1, n condominium@10, n family_home@10, n apartment@20;
}
```

since the comma after mansion@1 is interpreted to mean that the next item is another example of the same property.

The syntax for Declare commands is actually in the knowledge base itself. You can use the Giver of Names application to explore the knowledge base. If you start the Giver of Names application without a .script file in the Lexicon folder, you will see the GON Console window. You can type commands into it, and see the results. Typing the name of any entry displays the properties for that entry. For example, typing ‘computer’ and pressing the ‘enter’ key causes the following to appear in the console:

```
n computer@5.5
{
AgentOfVerb compute;
QualityImpartingVerb computerize;
Sense expert.computer@0.5;
Sense machine.computer@05;
}
```

The format is exactly like the Declare command without the Declare word at the start, except that the categories for the properties which are omitted for clarity, You can select any word in this text to explore further. Selecting machine.computer and pressing enter displays a much longer list of properties of _n (noun meaning) machine.computer with properties like Synonym, KindOf, Kind, and Part.

If you type Declare and press enter, you will see:

```
Command Declare
{
Template EntryList PropertySet;
}
```

Select EntryList and press enter to see:

```
GrammarElement EntryList
{
Template ThingRef, EntryList;
}
```

These “Template” properties show the syntax for the entry in question.

ThingRef does not have a template. It represents a single entry identifier, made up of a category and a name.

The comma after ThingRef in the Template has two meanings. It means that the comma and the rest of the Template is optional, but if you continue with an EntryList after the comma, the comma must be present. This nested template structure means that an EntryList can be made up of one or more ThingRef’s separated by commas.

PropertySet looks like:

```
GrammarElement PropertySet
{
Template { PropertyList };
}
```

PropertyList looks like:
GrammarElement PropertyList
{
Template PropertySpec (PropertyList);
}

The ‘{‘ at the start of the Template for PropertySet has a similar character to the comma seen above. It is optional, but if it is not there, the template ends right there and no PropertyList is accepted. If it is there, there must be a PropertyList and it must end with a ‘}’.

The parentheses ‘(‘ and ‘)’ around PropertyList are different. They say that whatever is inside them is optional but they themselves do not appear in the actual command.

PropertySpec is a declaration of a single kind of property. It has several possible Templates, but the key ones are described below:

Property

which expands to

ExistingRef InstanceList

ExistingRef is an entry that already exists in the knowledge base.

InstanceList expands to one or more Instances of the property.

The structure of an Instance of a property depends on whether the property has a template or whether it is defined to have a certain kind of value.

For example, a Colour property has the following Template

Template Hue Chroma Luma;

The Template property states that Colour properties have 3 parts. Looking into Hue, we see that it has an AtomType of Phase. Chroma and Luma have AtomTypes of Range.

Phase and Range are both internally defined as being expressed as a floating point value representing the value, and a tolerance which expresses how important it is that that component of the colour be close to the value. The larger the tolerance, the less important it is to the property.

InstanceList, in the case of a Colour property therefore expects 3 floating point values with optional tolerances. For example:

Colour 0.5@0.1 0.2@12.0 0.75@1.0;

NOTE: all Property declarations end in a semi-colon.

A Property can optionally be followed by a colon the word Consequence and a PropertyList of the actual consequences.

Sense
Sense properties define specific meanings of a word and have the following syntax:

Sense SenseList;

Sense is a literal term and the SenseList following the word sense has the template:

ThingRef (PropertySet)

The brackets around PropertySet mean that that section is optional. So a Sense property includes an Entry pointing to the meaning entry for that sense, and an optional list of properties specific to that sense of that word (most likely syntactic information).

EntryList
A property can simply be identified by name. So, for example, a preposition may be ‘exhaustive’ (meaning that it does not require any other words) like ‘outside’, and this property would be indicated in the list of properties as

prep outside{ Exhaustive; }

The template is Entrylist, because one or more of these simple property declarations can be combined into a single line separated by commas.

Sublist
A property can have a sublist that defines it. The syntax is:

Sublist PropertySublist;

and PropertySublist has the folowing template:

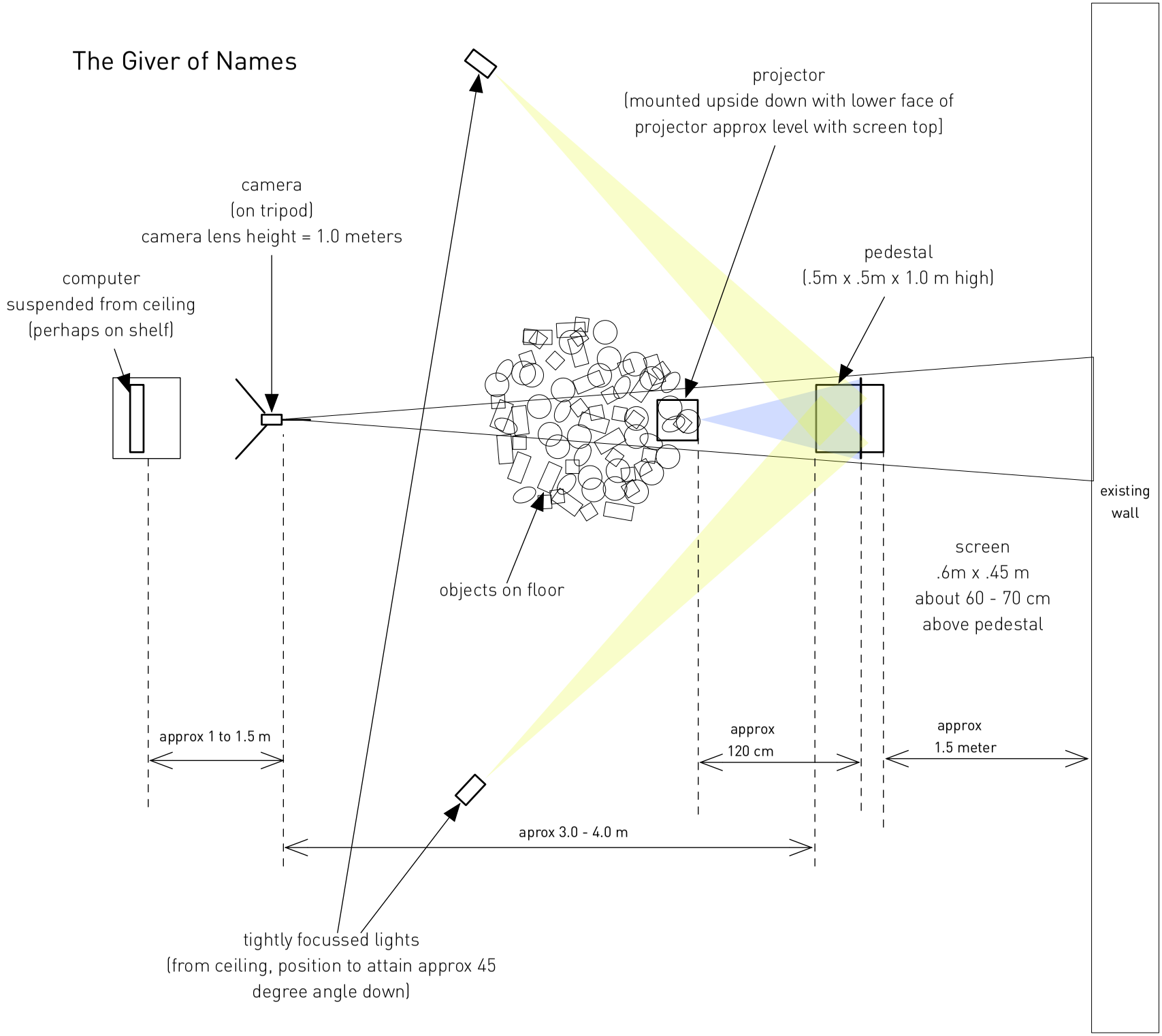
ExistingRef PropertySet;

See Sense above for more info on this template.

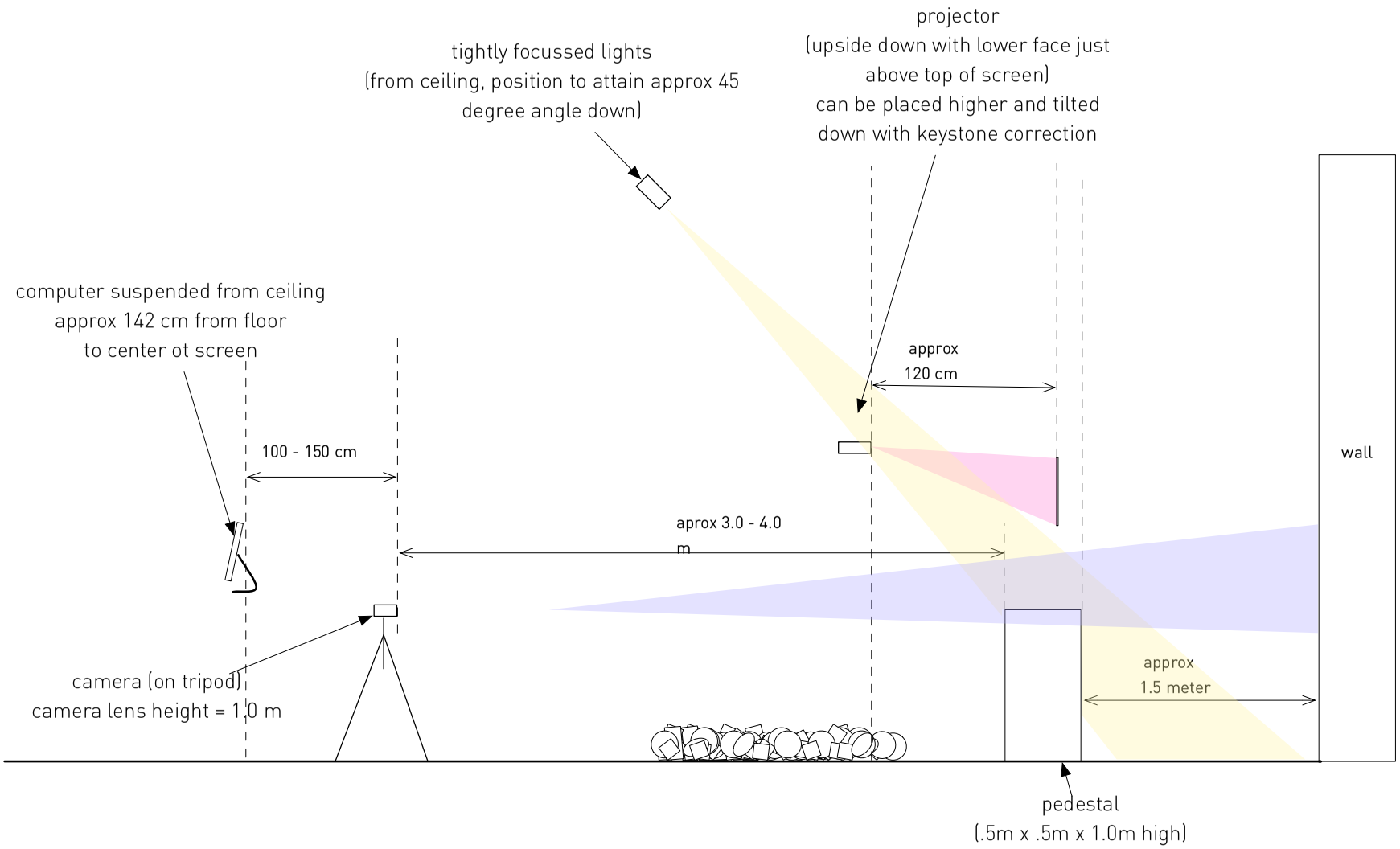
The other possible templates are more rare and can be explored in the knowledge base.

Appendix A: Drawings

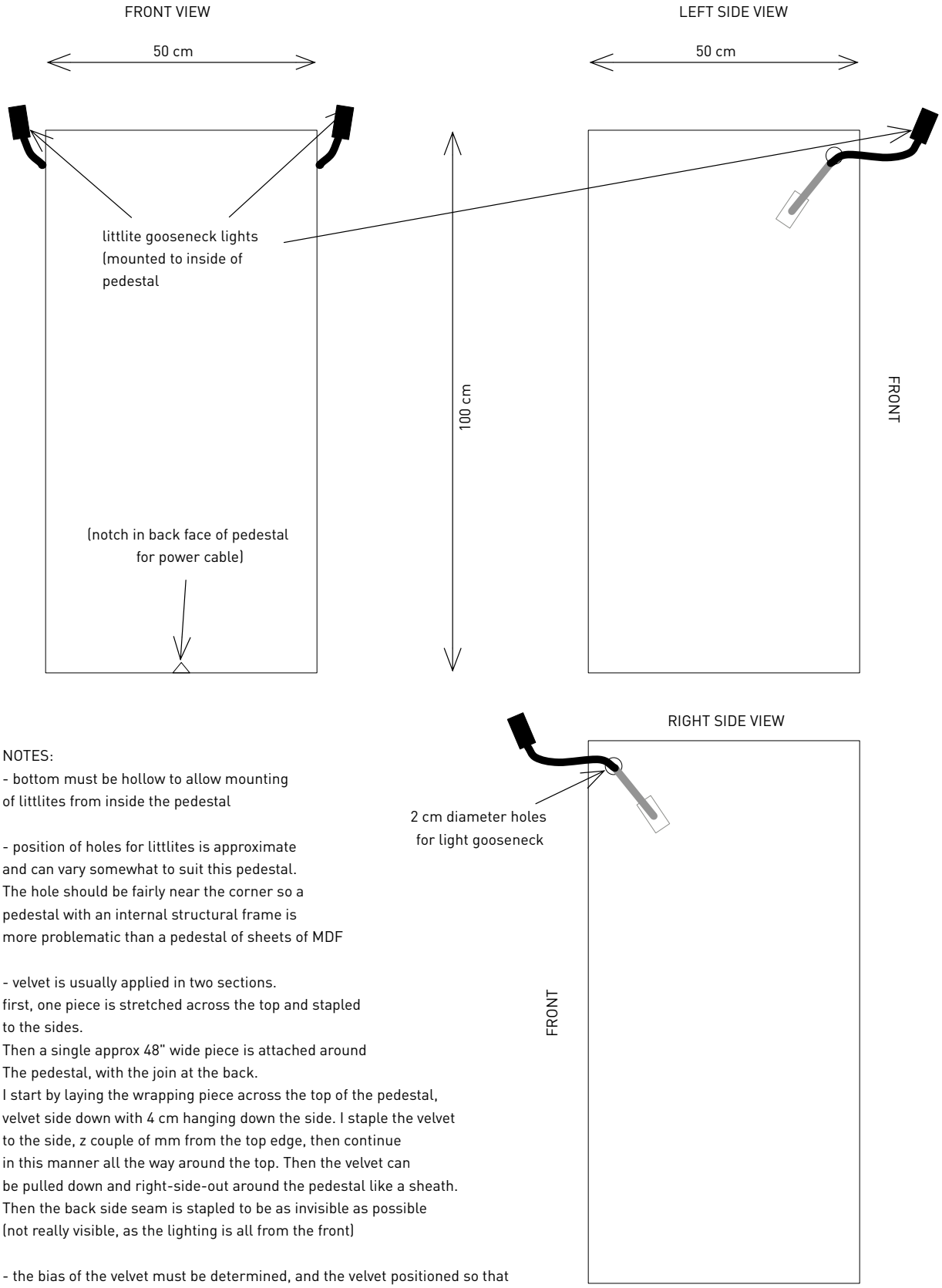
The Giver of Names



The Giver of Names : Side View



PEDESTAL - THE GIVER OF NAMES (DWG 1 Nov 3/08)



NOTES:
- bottom must be hollow to allow mounting of littlites from inside the pedestal

- position of holes for littlites is approximate and can vary somewhat to suit this pedestal. The hole should be fairly near the corner so a pedestal with an internal structural frame is more problematic than a pedestal of sheets of MDF

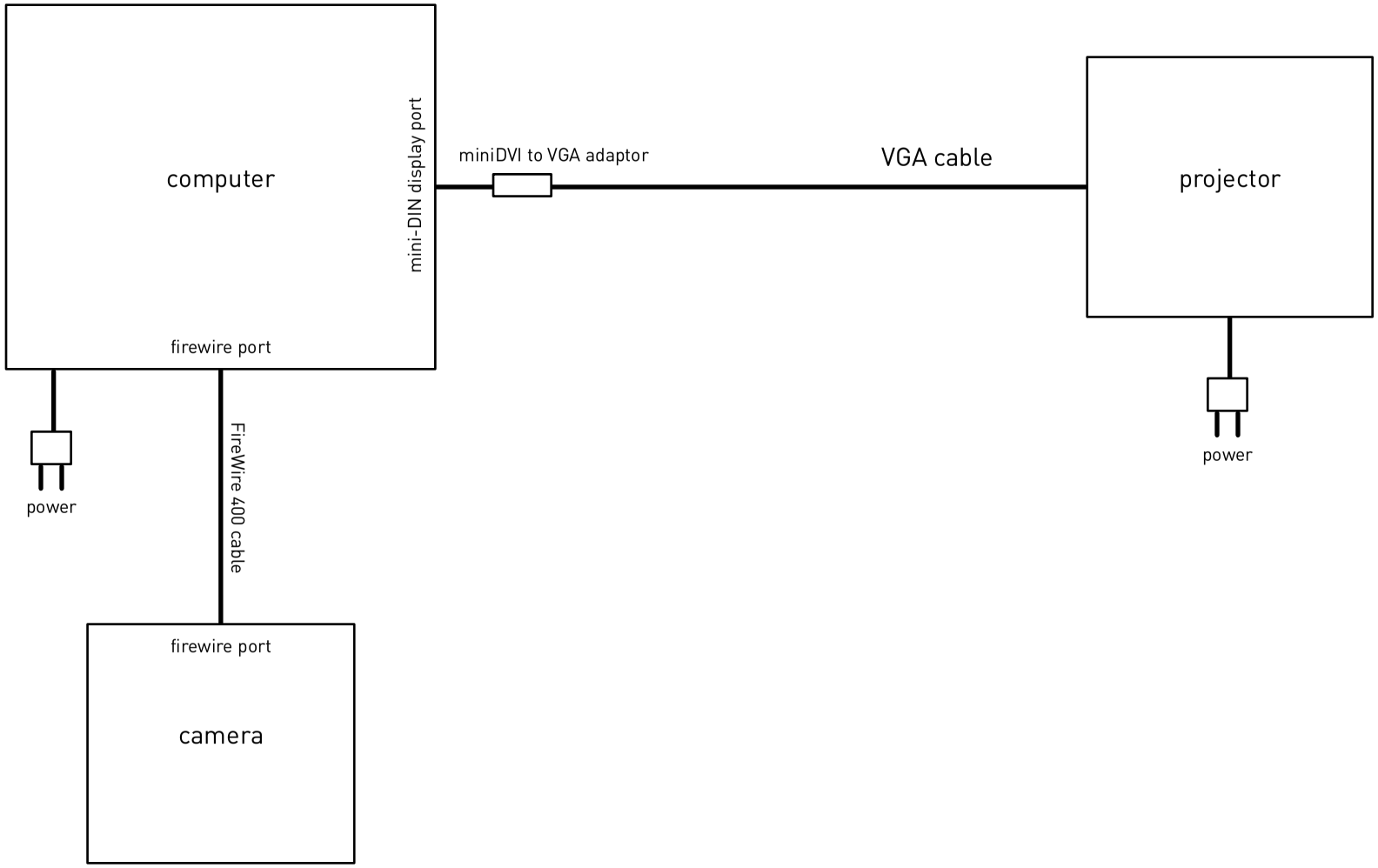
- velvet is usually applied in two sections.
first, one piece is stretched across the top and stapled to the sides.
Then a single approx 48" wide piece is attached around The pedestal, with the join at the back.
I start by laying the wrapping piece across the top of the pedestal, velvet side down with 4 cm hanging down the side. I staple the velvet to the side, z couple of mm from the top edge, then continue in this manner all the way around the top. Then the velvet can be pulled down and right-side-out around the pedestal like a sheath. Then the back side seam is stapled to be as invisible as possible (not really visible, as the lighting is all from the front)

- the bias of the velvet must be determined, and the velvet positioned so that the top and front are as black as possible from the camera's view point.

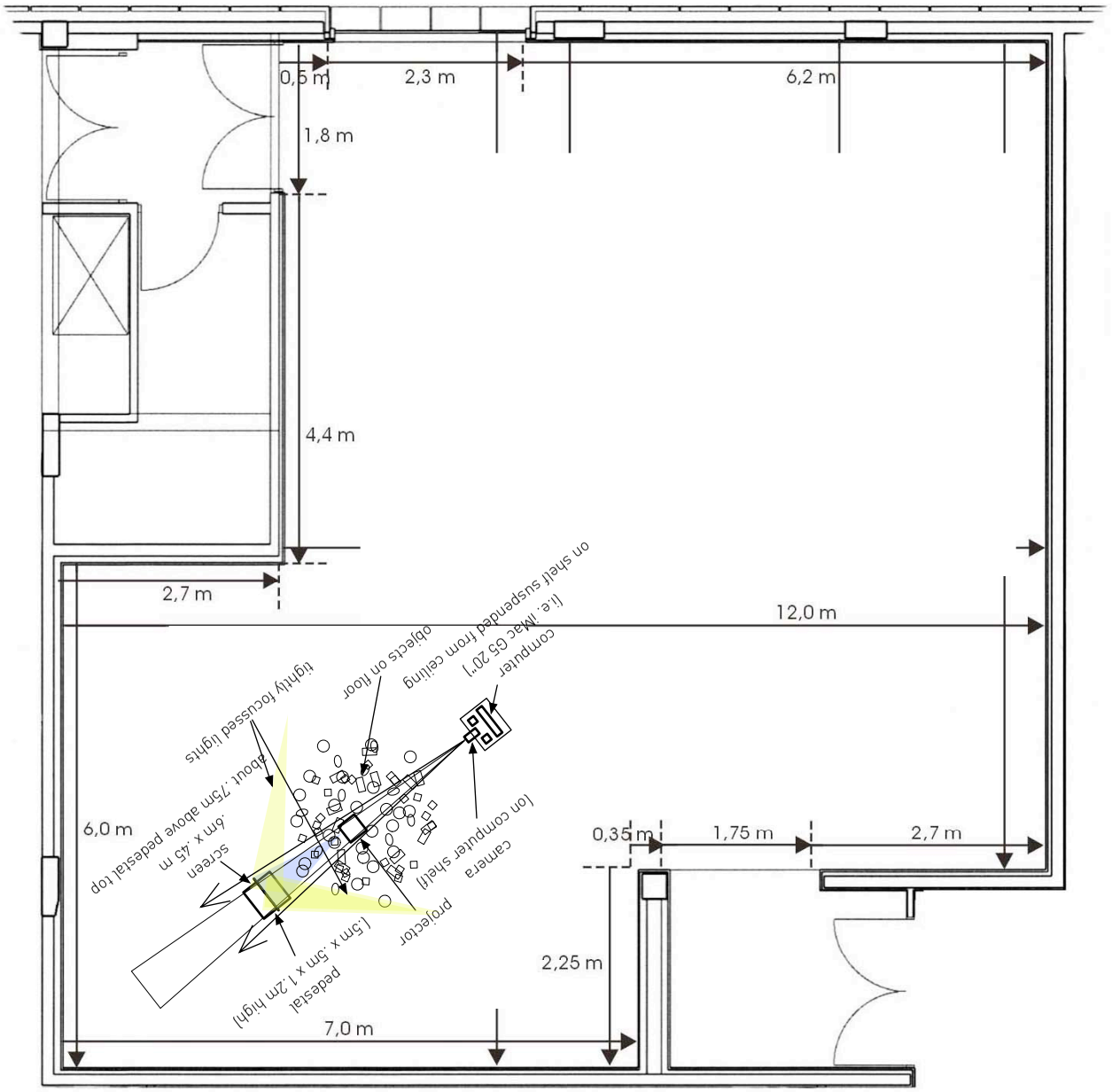
SCREEN - THE GIVER OF NAMES

NOTES:
- screen material can be any robust flat paintable material, thickness variable depending on material (perhaps 1/8" - 1/4")
-front surface should be painted approx 50-60% grey matte. (darkness of grey depends a bit of projector)
-back surface can match front colour, for simplicity.
- holes in top are for hanging.
- hanging options should allow for easy leveling (turnbuckle / aircraft cable?)
- monofilament is an option but a little harder to level...

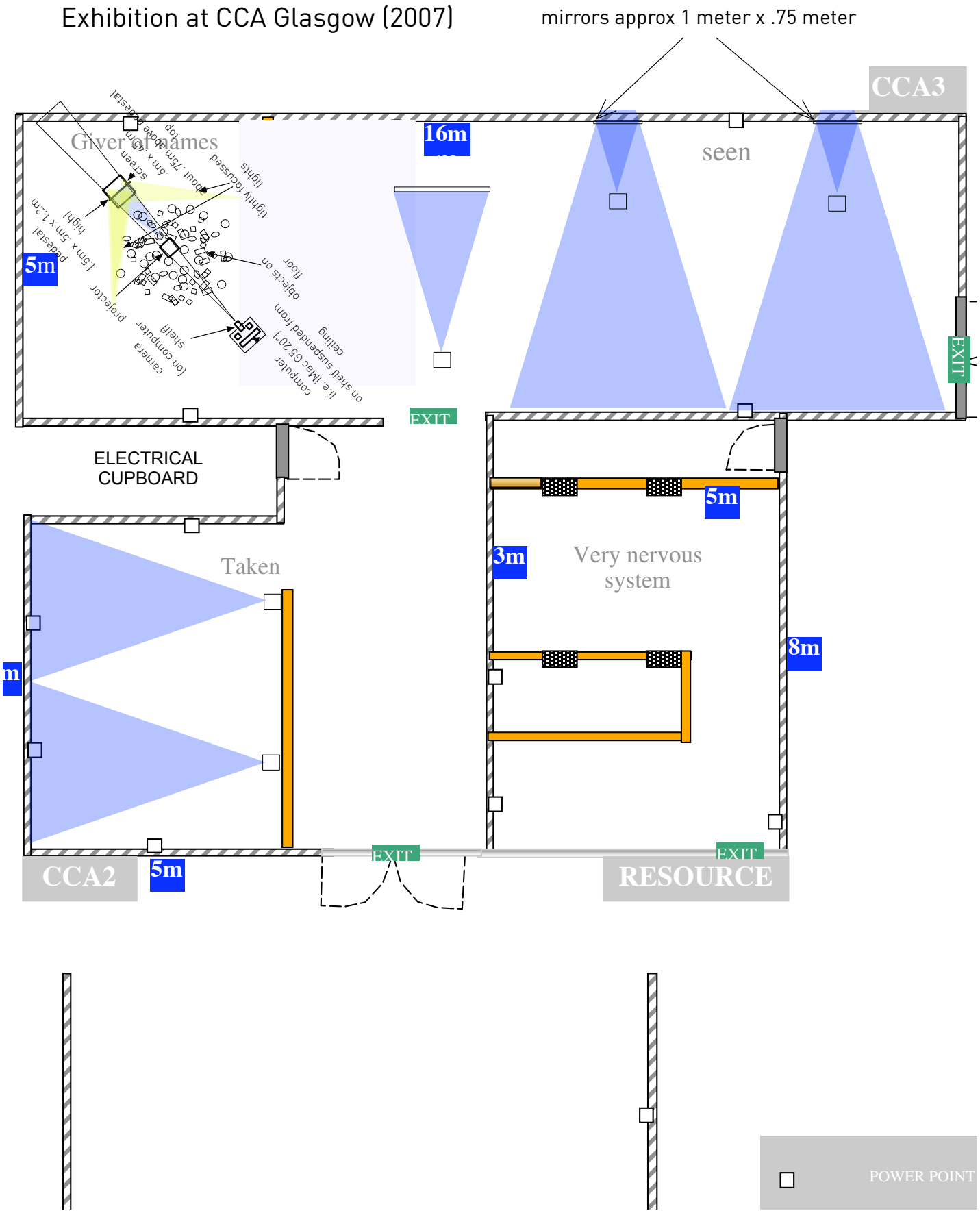
The Giver of Names
Connection Diagram

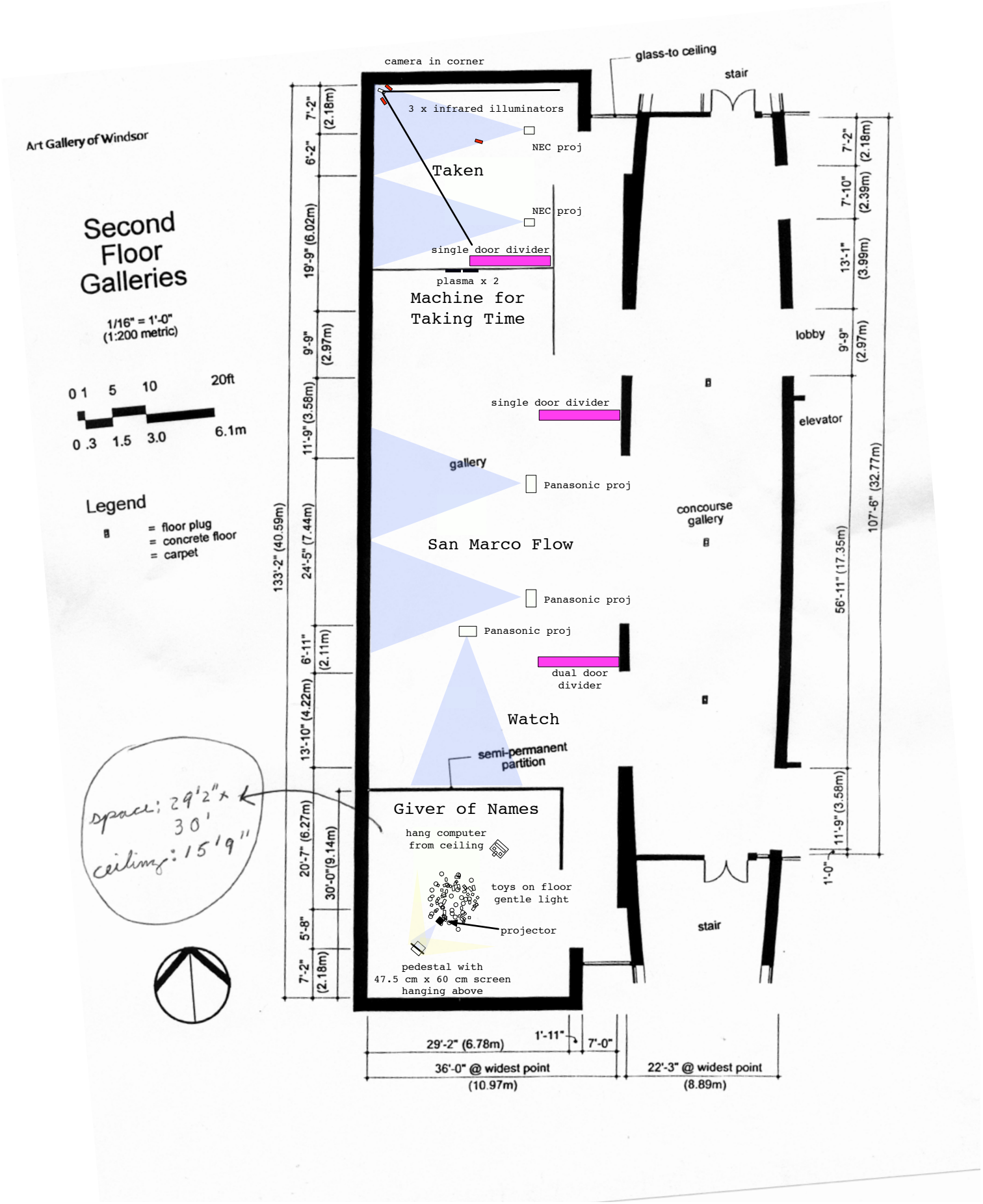


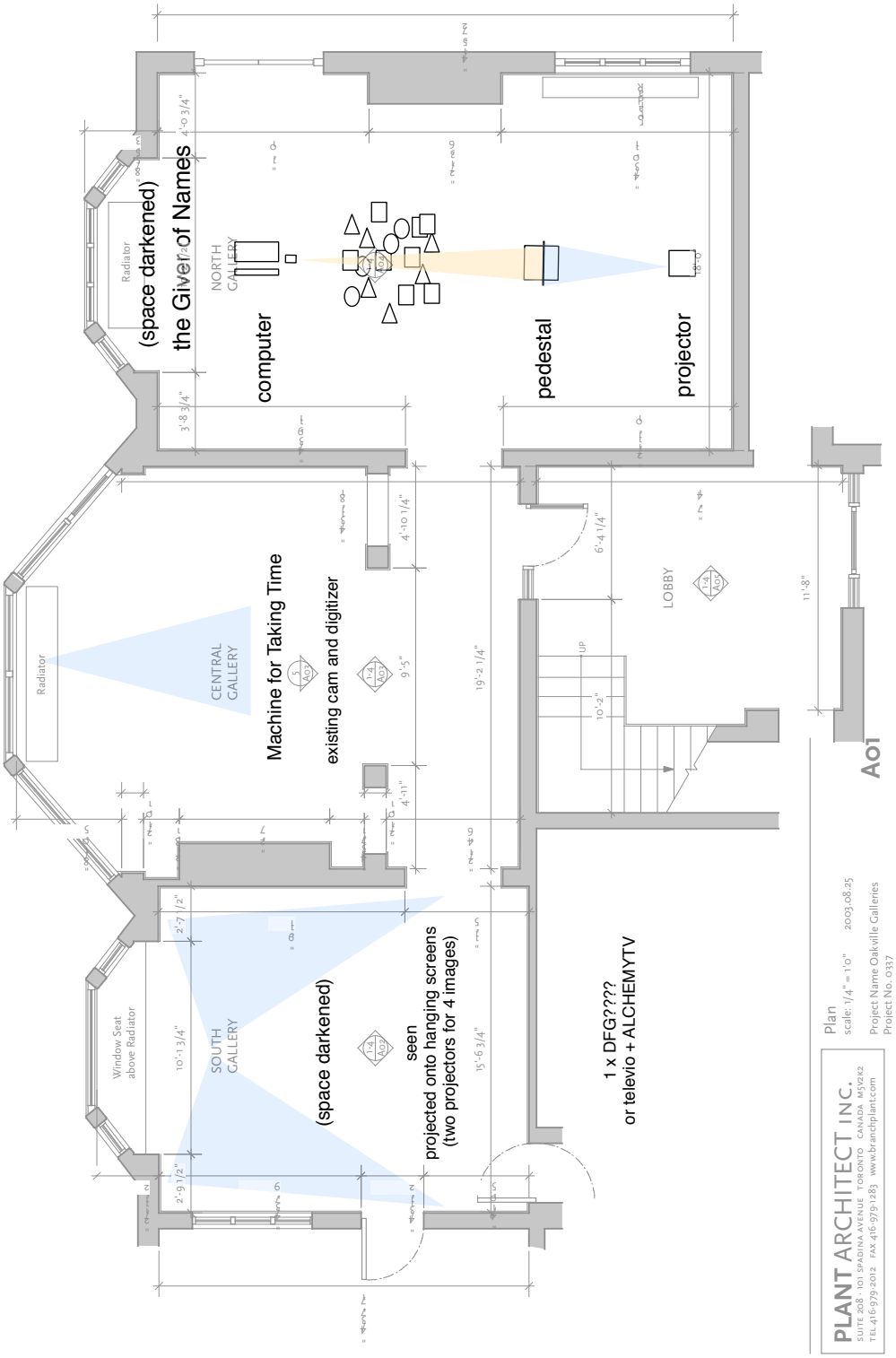
Exhibition at FACT, Liverpool, UK (2007)



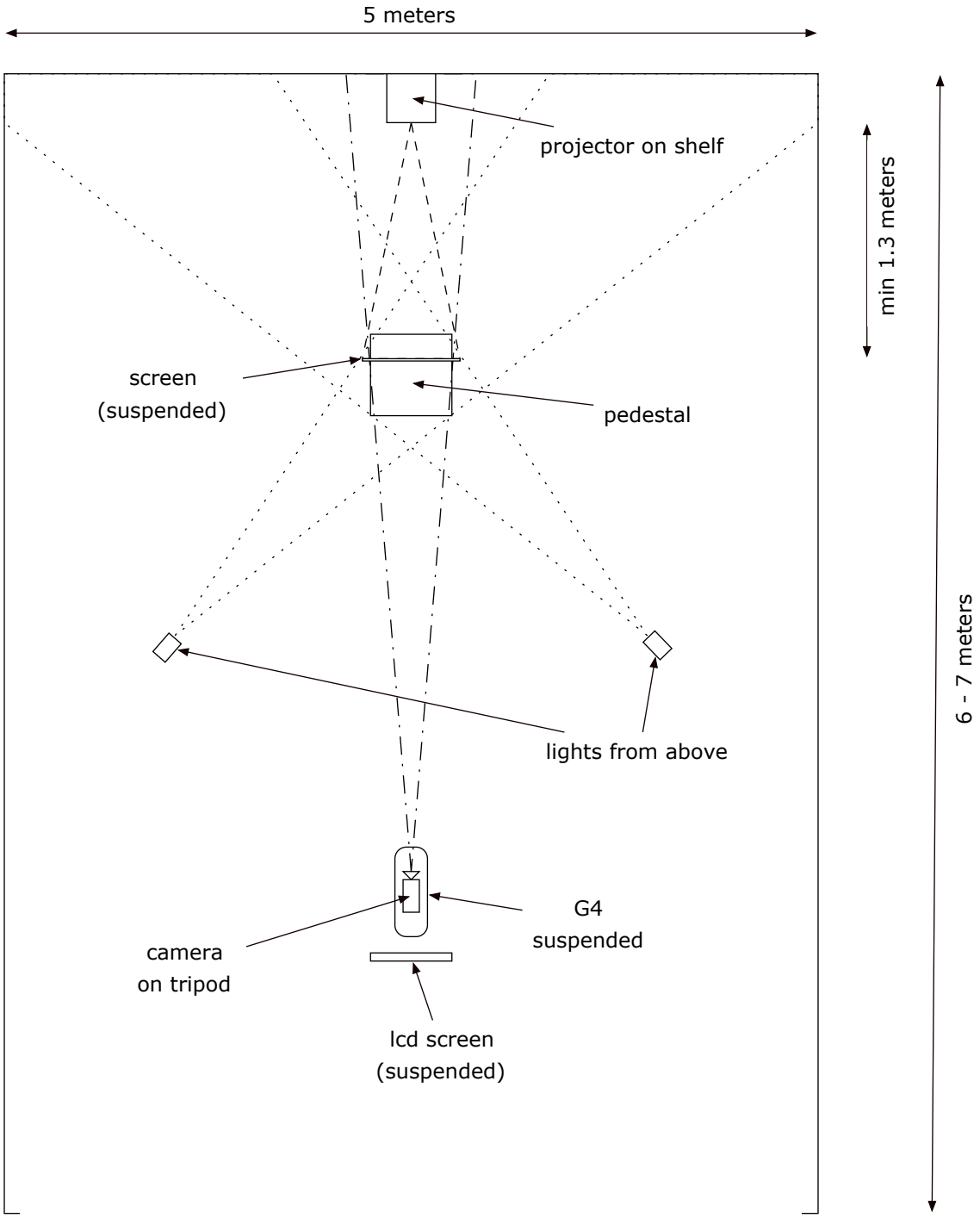
Exhibition at CCA Glasgow (2007)




Art Gallery of Windsor
Second Floor Galleries
1/16" = 1'-0"
(1:200 metric)
0 1 5 10 20ft
0 .3 1.5 3.0 6.1m
Legend
= floor plug
= concrete floor
= carpet
camera in corner
3 x infrared illuminators
NEC proj
Taken
NEC proj
single door divider
plasma x 2
Machine for Taking Time
glass-to ceiling
stair
lobby
elevator
single door divider
gallery
Panasonic proj
concourse gallery
San Marco Flow
Panasonic proj
Panasonic proj
dual door divider
Watch
semi-permanent partition
Giver of Names
hang computer from ceiling
toys on floor gentle light
projector
pedestal with 47.5 cm x 60 cm screen hanging above
stair
space: 29'2" x 30'
ceiling: 15'9"
7'-2" (2.18m)
6'-2"
19'-9" (6.02m)
9'-9" (2.97m)
11'-9" (3.58m)
24'-5" (7.44m)
6'-11" (2.11m)
13'-10" (4.22m)
20'-7" (6.27m)
30'-0" (9.14m)
5'-8"
7'-2" (2.18m)
133'-2" (40.59m)
7'-2" (2.18m)
7'-10" (2.39m)
13'-1" (3.99m)
9'-9" (2.97m)
56'-11" (17.35m)
107'-6" (32.77m)
11'-9" (3.58m)
1'-0"
29'-2" (6.78m)
1'-11"
7'-0"
36'-0" @ widest point
(10.97m)
22'-3" @ widest point
(8.89m)


(space darkened)
the Giver of Names
NORTH GALLERY
computer
pedestal
projector
Radiator
CENTRAL GALLERY
Machine for Taking Time
existing cam and digitizer
LOBBY
UP
Window Seat above Radiator
SOUTH GALLERY
(space darkened)
seen
projected onto hanging screens
(two projectors for 4 images)
1 x DFG????
or televio + ALCHEMYTV
Aoi
Plan
scale: 1/4" = 1'0"
2003.08.25
Project Name Oakville Galleries
Project No. 0337
PLANT ARCHITECT INC.
SUITE 208 · 101 SPADINA AVENUE TORONTO CANADA M5V2K2
TEL 416-979-2012 FAX 416-979-1283 www.branchplant.com

Art In Output Eindhoven (2003)



Ideally, the space would have an open front, and be painted black or dark grey inside

the Giver of Names proposed layout

Jan 21/03

Notes on some example exhibitions

Oboro (1998)
The space was long. I set the camera and pedestal far apart and put the display at the third point of a triangle. This was not an ideal layout. The work was too dispersed. The projection had not yet been included.

Kiasma (2000)
The space was large, but not too large, perhaps 9m x 9m. This was the firsts instance with the projection. We used rear projection with a very high quality screen and the results were good. I had lights on stands providing the underlighting. These were effective and dramatic, providing good shadows, but were too often knocked over by visitors.

Agnes Etherington Arts Centre (2000)
Similar setup to Kiasma. The space felt large but good.The camera tripod was well weighted down to prevent it from moving.

National Gallery of Canada (2002)
The space was huge (perhaps 12 m x 12 m) which felt too large. I spread out the installation to fill more of the space, but perhaps took it too far.

Art In Output (2003)
The space was a box built into a much larger public space. The space was narrow. The walls were black, but in this case, unlike many others, this worked okay, perhaps because it provided extreme focus, and one side was open to an air space, so claustrophobia was not a problem.

ZKM (2004)
The space was a 7 x 7 meter square room with a narrow doorway. The display was hanging at the front a few feet inside the doorway with the pedestal straight across the room. This was a bit awkward as a layout, but the room, once you were in felt okay.

Oakville Galleries (2004)
The entry to the space from along the long side of a rectangle. Therefore as you entered, you looked into the middle space with the toys on the floor. The pedestal and projection were to one side, and the computer and display were to the other. The display was mounted perpendicular to the line between camera and pedestal which worked for this configuration.

FACT (2007)
The camera was mounted on the underside of the computer shelf and the distance from camera to pedestal was at its minimum. This space was really a bit tight for the Giver of Names. The walls were painted black which was in retrospect, a mistake, making the installation too dark and slightly claustrophic. The approach was also a bit too constrained. It is nicer to be able to approach from an angle.

CCA (2007)
The camera was mounted on the underside of the computer shelf. The space was large and white. The Giver of Names felt airy, and welcoming. The installation was probably in the end slightly more spread out than in the drawing.

Montréal Museum of Fine Art (2007)
The space was painted dark blue which felt too dark. As opposed to FACT and CCA, the line from camera to pedestal ran straight across the space, not diagonally. The space felt a bit small. On entrance, you encountered the computer screen too directly.

AGW (2008)
The camera was mounted on the underside of the computer shelf. This space was a good size. The walls were white. It was nice that the initial view was only of the pedestal and toys and projection, but the computer screen felt too much in the corner and out of the way. In retrospect, I would have considered having the monitor facing towards the entrance, rather than back towards the corner

Campbelltown Art Centre (2008)
The space was a square, slightly small and painted white. The wall on the open side was open from the corner to the mid-point. The work was positioned straight across the room. Normally in a room this size, I would orient on the diagonal, but in this case, there was some light leakage into the space that slightly washed the corner that would have ideally had the projection screen and pedestal, so I rotated it to be straight across.

Appendix B: The Giver of Names Core

cKnowledge Detailed Introduction

The knowledge base is made up of tens of thousands of individual entries. Each entry has a small set of standard data members and an extensible list of properties, behaviours and interconnections.

Each entry has a category. Categories include:

category of english words:
`kVerbType`, `kNounType`, `kPronoun`, `kAdjectiveType`, `kAdverbType`, `kDeterminer`, `kConjunction`, `kPreposition`, `kInterrogation`

category of word related meanings:
`kNounMeaning`, `kVerbMeaning`, `kAdjectiveMeaning`, `kAdverbMeaning`, `kPrepositionMeaning`

punctuation:
`kPunctuation`

ideas and properties:
`kConceptType`

various internal administration types:
`kFileType`, `kInternalProperty`, `kPlaceHolder`

groups holding a group of entries sharing a name but with different types
`kGroupType`

elements used in defining command grammar
`kGrammarElement`, `kGrammarMessage`, `kGrammarLiteral`

grammar expansion trees
`kTreeType`

commands:
`kCommandType`

the categories themselves:
`kCategory`

Each entry has an identifying ID number which is used to identify a specific entry internally. All entries in the Knowledge-base are also accessed by name and category. Names are case-sensitive. Where multiple entries have the same name, but belong to different categories, the name intially points to a kGroup entry, which contains a list of entries that share that name. These groups are maintained internally and are not reflected in the text form of the database.

English word entries and their syntactic behavioural properties are separated from meaning entries. Meanings point to the words that are used to express them with “Synonym” properties. Words point to their various meanings with “Sense” properties.

For example, in the language of the database:

```
Declare n horse
{
Sense _n equine.horse;
SoundsLike adj hoarse;
}

Declare _n equine.horse
{
Synonym n horse;
}
```

(n means kNounType, _n means kNounMeaning)

Concepts hold a wide variety of properties and characteristics like “Colour”, settings parameters, relationships like “KindOf”, and anything else that is neither word nor meaning, but that does not fit the stricter definitions of the other types. By convention, concepts have initial capital letters, whereas words do not.

The grammar elements, messages and literals are special terms that can be placed in templates that define syntactical structure for commands and for english sentences. They are used to update various internal states of the parsers or expanders during processing. The templates themselves are used to define the syntax for commands, knowledge declarations and the expansion syntax for grammar trees.

In their entry records (of class cKnowledgeEntry) they contain standard entry data such as category, entry weight, various flags, a pointer to a property table if it exists, etc.

Each entry optionally points to a list of properties and connections with other entries. The lists are made up of strings of cAtom objects and are stored in cLists objects.

cAtom
cAtoms are the lowest level of knowledge data.

A full list of atom types is presented in Appendix D. In general, atoms contain values, refer to other entries directly, express a relationship with another entry, claim or declare a property, point to strings or property structures or templates. Property atoms are a class of atoms that express properties. A property atom has a property ID and some sort of expression of the property attributes, which could be a value, another entry, or an index to a record in a table of more complex properties.

In addition to their “cKnowledgeEntry” records and attached property lists, entries have a stimulation state indicating how strongly stimulated this entry is based on recent input (usually from the vision system).

Navigating Lists

While an entry owns one list, this list may be made up of multiple fragments. A cListMarker is an object that navigates along a single list, retrieving individual atoms, appending to a list, etc.

For many reasons, it is often desirable to navigate an entry’s list to a greater depth, stepping into properties and related objects along the way. This is effected with a cKnowledgeGroup, which is an array of cListMarkers. When and where this navigation steps deeper is defined by a scope object of class cScope. By way of example, a cScope object can be set up to cause the cknowledgeGroup object to step into all “KindOf” properties, so that the entry effectively inherits all properties in the lists of any entries up an inheritance tree of “KindOf” properties. For example, “equine.horse” is a kind of “odd-toed_ungulate.equine” which is a kind of “ungulate.odd-toed_ungulate” which is a kind of “placental_mammal.ungulate” which is a kind of “mammal.placental_mammal” which is a kind of “vertebrate.mammal” which is a kind of “chordate.vertebrate” which is a kind of “animal.chordate” which is a kind of “life_form.animal” which is a kind of “entity.life_form” which is a kind of “Noun.entity”. Opening the “KindOf” scope for a cKnowledgeGroup causes the list for the entry “equine.horse” to include all properties of all the entries up the “KindOf” chain. Various forms of property expansion are possible by setting parameters of cScope. Similarly, opening the “Kind” scope would open the list to include all properties of the sub-categories in the “Kind” chain. This tool allows for selective partitioning of the database for various functions. For example, one might open the “Kind” scope, then use a cKnowledgeGroup to iterate through an entry list, to gather all “Synonym” properties. This newly constructed list could be used to narrow a fairly abstract notion like “mammal” to a specific animal. This process of choosing from this list could be weighted by the stimulation level of each specific animal’s entry.

Stimulation

Each entry also has a record holding information about the state of stimulation of that entry. Entries are stimulated based on input from the vision system. The stimulation spreads through the knowledge base from entry to entry via the properties in the entries’ lists.

When a quality of part of the image is determined, that quality is stimulated, meaning that the software first increases the stimulation level of this quality's entry, then walks along its list, stimulating the entries representing each claimed property in turn. The state of the cScope object determines how far the stimulation spreads beyond the core list of the entry.

Certain kinds of properties can also pass stimulus back to all things that claim that property. Some properties are declared as having a reverse index. Every time an object declares this kind of property, the declaring object is added to the list. Through this mechanism, the noun "orange" (as in the fruit) would stimulate its declared colour property of "orange" and through the colour orange's reverse index would stimulate all other entries that declared the colour property "orange".

Long and short term stimulation levels are maintains in the stimulation array. The actual stimulation level for an entry is its short term stimulation attenuated to some extent by its long term stimulation, in order to make the system more responsive to novelty, and to keep it from getting "obsessed".

The total "landscape" of stimulation for the entire knowledge base is the context in which the Giver of Names formulates its sentences. These levels influence its choices as it constructs the sentence by expanding its grammar trees.

The knowledge base also defines the syntax of a command parser, and the expansion trees of a model of english grammar.

The Class Hierarchy of cKnowledge

cBlock : a block of memory

cTableArray : a cBlock containing an array of elements of fixed size

cListTable: a cTableArray that owns a cLists object and a cMarkerSet for navigation.

cIndexedListTable: a cListTable that allows for a reverse index for each entry. This is usually used in property tables where a particular colour might list all objects that have claimed it as a property.

cRecordListTable: a cIndexedListTable which treats the array elements as records. Actually, the element starts with an atom pointing to the associated list. The rest of the element is treated as a cAtom-based list of fixed length.

cKnowledgeListTable: a cRecordListTable including a cKnowledgeGroup list marker structure for cKnowledge specific navigation.

cKnowledge: a cKnowledgeListTable with a file of entry names, a hash table for rapid access via name, a stimulation array to hold stimulation data for each entry, an index table, a command parser, some extra markersets for searching lists and other functions, and an articulator for formulating sentences.

cArticulator

The cArticulator class manages the language generation for the Giver of Names. It depends on syntactic and semantic information in cKnowledge. It builds sentences using grammar rules held in cKnowledge and fills the various slots in the sentence structure from words held in cKnowledge, often chosen because of their level of stimulation.

cArticulator's grammar is defined largely by "Tree" objects in the knowledge base. A "Tree" object generally has one or more template properties which define how a Tree expands. For example:

```
Declare Tree DeclarativeClause
{
Template Subject VP PP;
}
```

The cArticulator object evaluates each element in the template, expanding further into deeper templates where appropriate and making decisions where options exist based on weighted random processes and the stimulation levels in the knowledge base. Eventually, all the positions in the tree are filled with layers of clauses, modifiers, and

ultimately single words. Once the entire tree is expanded and resolved, the sentence an be constructed. Rules of subject / verb agreement are handled in code, which verb exceptions are defined in the knowledge base. The syntactic information used by cArticulator is augmented by semantic hints held in the knowledge base. An entry for a specific verb might specify one or more acceptable subject options which might be specific words, or very high level concepts like “entity.life_form”. In addition, the knowledge base includes lists of appropriate choices based on the parsing of a large number of texts. For a verb entry, there will often be a series of “T_Subject”, “T_Object”, and “T_ModifierPrep” properties. The “T” properties are all generated by the automatic parsing of existing texts.

When cArticulator chooses a word for its sentence, it reduces the stimulation of the chosen word, and that stimulation reduction spreads through the properties in a similar way to the stimulation process.

In effect, a sentence starts with almost infinite possibilities within its riles of grammar, and it is cArticulator’s job to progressively narrow those possibilities until an actual sentence results. Where there is a reason to make a particular decision among the options, that decision is taken. Where there is not, the decision is made randomly, guided only by a “Verbosity” parameter that defines approximately how elaborate a sentence should tend to become.

cArticulator will sometimes fail, and generate a c++ exception. This prints an error message to the console, but causes no harm.

cArticulator’s main component is the cEnglishExpander.

cEnglishExpander

cEnglishExpander is manages the grammatical expansion and rendering into English of sentences. It can be divided into the sections that manage the various levels of a sentence: Clause, NP (Noun Phrase), VG (Verb Group), PP (Prepositional Phrase), and DG (DeterminerGroup).

The cEnglishExpander uses a structure called cEnglishTree to hold the state of the sentence as it expands. cEnglishTree is a kind of ListTable. The initial template from which the sentence is expanded is the list for the first entry in the listTable. When a token in the template expands into another template, the token becomes a LocalExpansion Atom that points to a new entry in the ListTable with the expansion template as its list.

At any point, the cEnglishExpander is always in a Clause node, and NP node, a VG node, a PP node or a DG node. Each distinct node has its own context which holds state variables or registers that define accumulated characteristics of that node. There are cClauseContext objects, cNPContext objects, cVGContext objects, cPPContext objects and cDGContext objects which are all sub-classes of cEnglishContext. A node may persist across several template expansions, in which case the the existing context

registers continue to apply. When a sub-node is completely resolved, the cEnglishExpander steps back down a level and continues processing any remaining tokens in the template. At this point the context returns to what it had been for that level.

Once the cEnglishTree is completely expanded and resolved, the tree is traversed again by the various node renders which translate themselves into english sentence fragments. When a clause node is being rendered, and an NP node is entered, the NP renderer takes over, renders the node as a string and returns the string which is inserted into the clause in the appropriate location. The result is a complete sentence.

In many cases a token does not simply expand into a template. A token may have a “Frame” property instead of a template. Frame properties point to entries of category kTreeType. A tree may contain one or more template properties, or one or more Frame properties that refer to other Trees. Just as with templates, when a token resolves to a tree entry, a new level of the cEnglishTree is added, new contexts are created if a change of node takes place, and the property list of the tree is navigated, and the properties evaluated. Most of these properties will define how certain registers in the current context should be filled. Any property ending in a capital R (i.e. SubjectR) sets the register in the context to the property value or reference.

Where multiple Frame properties exist in a property list, a choice is made about which to expand. Some Frame properties have specific consequences.

Example:

```
Declare v abide
{
Sense _v stay2.bide
        {
        Frame Tree HumanIntransitivePP;
                {
                MainPrepR _prep accompaniment_prep.with;
                }
        }
}
```

In the above case, the sense of the verb abide with the meaning stay2.bide has an expansion tree of type “ HumanIntransitivePP”, and carries the added consequence that the MainPreposition register of the PP context that will result is defined by the preposition meaning “accompaniment_prep.with”. In simpler terms, if you use the v abide in the sense of the stay with someone, then you must abide “with” that someone.

Trees may also define “Constraint” properties. These properties are most often found in “selector” trees. Selector trees are uses to choose a particular word, and the Constraints define necessary properties that that word must possess.

Example:

```
Declare Tree MannerAdverbSelector
{
Template ^ adv;
Constraint Frame Tree AdverbAfterVerb;
}
```

A manner adverb is an adverb used in this way:
He picked up the wallet casually.

The "^" token before adv in the template means to expand the following category to a particular entry of that category (i.e. pick an adverb). The Constraint property says to pick an adverb that claims a Frame property of Tree AdverbAfterVerb.

The final sentence that the cArticulator produces includes some inserted marker to indicate where a new line should begin ("_newline_") and where an indent is appropriate in the new line ("_indent_" or conversely "_unindent_").

When cEnglishExpander is facing a choice among options, it generally constructs a cChoiceTable object, fills it with the possible choices by navigating through the appropriate property lists and then makes a weighted random selection from the table using the property weights or stimulation levels to establish the probabilities.

cPerceiver

The cPerceiver controls the video source and analyses the images that it receives. Its initial tasks are:

1/ capture the image by taking 8 consecutive images and averaging them to reduce noise
2/ determine whether the pedestal is empty, or whether there was movement

If the pedestal is empty or there was movement, the image is discarded, and the process stops here.

3/ normalize the image to get optimal contrast
4/ find the significant edges in the image
5/ replace edgy pixels with the most similar non-edgy pixels in the immediate neighbourhood, to remove false colours that occur along edges.
6/ group pixels into areas of similar colour.

Then cPerceiver uses cPictureParser and cPictureTree to group the resulting patches of similar colour (called facets internally) into a picture description (cPictureTree). Groups of similar colour that are touching become "facets" of "objects", patches of a different

colour fromm a surrounding object become “contained objects”. Groups of differently colours objects that are touching form an object group.

Once the parsing of the image is done, the layers of objects in the picture tree are analysed to determine the colour of the object, its outline, its “moments” (symmetry, top-heaviness, etc), its size, and its position. The outline is calculated as a 2 dimensional FFT of the outline filtered to simplify the shape.

Once the image is parsed and analyzed, the Giver of Names goes through the tree and stimulates the knowledge base using these colour, outline, moment, size and location qualities. Colour and outline stimulation is done by looking for a colour or outline property in the colour and outline property tables that closely matches that of the object. When close matches are found, the stimulus form the object spreads through the inverse index of the property. Relative locations and sizes of neighbouring objects also trigger qualities of relationship like domination, leaning on, lying under, etc.

cAssociater

The cAssociater class was intended to coordinate communication between the cKnowledge, cArticulator and cPerceiver. In practice, these objects are often communicating directly with each other. cAssociater does control the initial reading of data and settings files, but mostly stays out of the way during actual processing.

Appendix C: The Giver of Names Command Syntax

Creating Entries

`Declare <list of entries> { >list of properties> }`
used to create an entry or add new properties to existing entry

`Edit <category> <entryName>`
prints a description of the entry in the console, in a format that can be edited and executed as a Replace command

`Replace <category> <entryname> { <propertyList> }`
Replaces the entire description of the entry with the supplied properties

`Clear <entry list>`
empties and entry of properties

Viewing Entries

`ID <entry ID #>`
'ID' command - display Entry for entry with specified ID

`Show <entry>`
OR
`<entry>`
'Show' command - display Entry for entry with specified name (and optional category)

Sentence Generation

`RandomSentence <optional list of givens>`
This generates and speaks a random sentence (using the current stimulation levels as a guide)
The Givens list is a set of sentence properties to enforce where possible in unfolding the sentence
For example one could put 'MainVerbR v help' in the givens list and 'help' would be extremely likely to be chosen as the main verb of the first clause evaluated.

`SayTree`
This displays the grammar structure of the previously spoken sentence

Camera Commands

`camera <parameter> <mode> <value(s)>`
This command sets the parameter of the camera to the mode and value
mode is optional if 'manual' and values are provided

mode can be auto (in which no values are required), manual (in which values are required), one_push (no values required), absolute (values are floating point) parameters are:

```
white_balance <mode> <float> <float>
white_shading <mode> <float> <float> <float>
brightness <mode> <float>
exposure <mode> <float>
sharpness <mode> <float>
hue <mode> <float>
saturation <mode> <float>
gamma <mode> <float>
shutter <mode> <float>
gain <mode> <float>
iris <mode> <float>
```

`camera report`
Prints the current settings of the camera parameters in the console window.

`LoadImage <path>`
Loads an image using the internal format

Export Commands

`Dump <path or filename>`
given a pathname of filename, dump the knowledge base to a .dump file

`ExportTags`
Creates a text file that contains all the tag properties in the current knowledge base

Settings Commands

`ExportSettings <path or filename>`
Exports basic GON settings for initialization of GON on startup as a .script file
Saves

```
BlackLevel
NoiseThreshold
MotionThreshold
BlackThresholdFactor
```

and the camera settings including:

```
exposure
white_balance
white_shading
brightness
```

```
exposure
sharpness
hue
saturation
gamma
shutter
gain
iris
```

`DelayScript <seconds>`
Delay the execution of a script file for the specified number of seconds after startup

`SetFont <fontName> <fontScale>`
Set the font for the displays with a relative scale (1.0 is standard... many fonts will be better at 0.6-0.7)

Entry Stimulation

`Stim <category> <entryname> <stimulation amount>`
Stimulates the entry corresponding to <category> <entryname> with the amount of stimulation indicated

`Deplete <category> <entryname> <depletion amount>`
Reduces the stimulation of the entry corresponding to <category> <entryname> with the amount of depletion indicated

`DepleteAll <optional depletion amount>`
Reduces the stimulation of all entries with the amount of depletion indicated (default is 0)

`MostStim <category> <sortDepth>`
A utility function to output the <sortDepth> most stimulated entries of <category> category

`SayStim <category> <name>`
Prints the current stimulation stats of this entry
output is formatted as
`name@dampedStim(recentStim/accumulatedStim)`
dampedStim is calculated by attenuating recentStim by accumulatedStim (if continuously stimulated, the system gets bored of that entry)
dampedStim is the stimulation value used by GON most often

`ShowAllStim`
Display the current stimulation of all stimulation contexts in the display window as a stimulation image

`AccumStim`
Accumulates the stim in all contexts into the global stim array context

`StimIndex <propertyname> <entryname> <stimAmount>`
Go through the index for the entry and stimulate any index atoms of propertyname type

`StimColour <name of entry with colour property> <stimulus amount>`
Find each colour property of this entry and stimulate anything in the index of colour property instances that are a reasonable match with the colour

`StimHSI <hue> <saturation> <intensity> <stimulus>`
Stimulate entries in the indices of colours in the colour table that are close matches to this one

`StimOutline <amp1> <amp2> <amp3> <amp4> ... <amp42>`
Stimulate entries in the indices of the outlines in the outline table that are close matches to this one

`MostStimmedDescendant <category> <entryname>`
returns the most stimulated descendant of the specified entry as
`<entryname>@<dampedStim>`
`SetContext <context number>`
sets current context (there are several contexts in cStimulationArray)

Knowledge Base Manipulation Commands

`LinkGroups`
Utility routine to link words of different type but same name (implemented in cKnowledge as groups) with SameWord property links to each other

`MostTags <category> <sortDepth> <maxTags>`
A utility function to determine the most tagged words in the knowledge base

`CheckOutline <42 floats defining harmonics of outline>`
This shows which outlines in the outline property table are close to this outline

`CheckColour <hue> <saturation> <intensity> <stimulus>`
Prints a list of colours that were stimulated by the supplied colour

`Parent <list of entries>`
This prints the closest parents shared by all the members of the list

`Generalize <entrylist>`
This attempts to work out the best generalization for the entries in the list

`GeneralizeProperty <entry> <property>`

This attempts to find optimal generalization for the property values of property type for the entry. For example, this would try to come up with reasonable higher level nouns to encapsulate a maximum of nouns in the list. So if the Property being generalized is T_PreferredObject and the entry is v eat, then the list will presumably mostly include foods, and the generalize command should find that most of the preferred objects could be replaced by the idea of food.

This function does not change the source list... it only prints out the results of the calculation.

`Contact <entry1> <entry2>`
tries to find the closest link between two entries in the database

`HSI <red> <green> <blue>`
This utility command outputs the hue saturation and intensity values that match the rgb of the supplied colourIn both standard HSI and my internal HSI implementation

Appendix D: Atom Types

kEntry: points to another entry by ID. (i.e. used in kGroup list to point to members of the group.)

kListReference: points to the beginning of a list of cAtoms in a cLists object.

kGroup: points to a list that contains a list of the group members

kStringReference: points to a text string

kValue: an integer value

kFloat: a float value

kHeader: an atom that holds the header information for a string of cAtoms.

kLocalExpansion: points to an entry and their sublist

kChoiceList: points to a sublist which contains a weighted list of options

kProperty: claims a single property by ID

kPropertyRecord: points to a property record in a property table at specified index.

kRange: contains a float value and a tolerance

kPhase: contains a float value and a circular tolerance (i.e.for hue information that loops from maximum to zero)

kLongValue: contains a 32 bit integer

kInheritance: points to an entry that this entry directly inherits from

kSense: points to a particular meaning of a word and optional points to a sublist of properties specific to that sense in the context of this word

kPropertyStructure: points to s sublist that contains a list that represents a property's multi-atom structure.

kPropertyIndirect: refers indirectly to an instance of a kind of property by pointing to an entry containing such a property

kRelation: expresses a relationship of a certain type between this entry and another entry.

kPropertyString: points to a sublist that holds a string for the property

kIndexOfProperty: refers to the index list of an entry (the list of all entries referring to this entry)

kConstraint: points to a set of constraints that a certain grammar tree choice imposes.

kConsequence: points to immediate ramifications of the choice of a particular Tree. Generally used to define the kind of preposition allowed after a verb

kPropertySublist: points to a property expressed as a sublist of atoms.