

The Helga Specification

Version 1.0

January 2009



© 2009 Brian Cipriano

This document is released under a CC-BY-ND license, version 3.0
<http://creativecommons.org/licenses/by-nd/3.0/>

Audience

This document is intended as an introduction to the philosophy and key concepts of Helga. You do not need to be a programmer or be technically fluent to understand this document. As we touch on each subject, we'll explain the subject as well as outline some basic guidelines that each implementation of this Specification should follow.

This document will be helpful to all new users of Helga, though it is not at all necessary to read it in order to use Helga. Current Helga users will benefit from reading this document as well; there's plenty to learn about Helga that might not be apparent through everyday use. This document is a must-read for Helga developers: we recommend that any aspiring Helga developer start by reading this document.

Some subjects will very rarely be encountered by the majority of Helga users; these sections are in italic and begin with the words "*Developer's Note*". If you do not plan on developing Helga tools, feel free to skip these sections.

Preamble

Helga is an open-source software project that facilitates the production of digital art, particularly video and animation productions. The heart of Helga is a production management system that organizes all data in your Helga install - multiple projects, shots, scripts, users, or anything else, arranged the way you want it.

Helga unites artists and engineers by providing one environment in which both arts production and tool development can function. Helga applies successful practices of open-source development to both, allowing long-distance, distributed work and subsequent peer review. Helga provides easy communication pathways to both groups, encouraging collaboration.

Helga does not aim to create an all-inclusive set of software tools for artists, though many are available. The main focus of Helga is to create a robust standard through which new tools can be created for your install and shared (if desired) amongst all Helga users. Tools will be available for command line use, web use, or both.

A Helga "site" is comprised of four things:

1. Your Helga data - project files, production support scripts, and a database storing asset metadata.
2. HelgaShell - a command line interface to your Helga data.
3. HelgaWeb - a web interface to your Helga data. In order to minimize code repetition and ease maintenance, HelgaWeb should make calls to HelgaShell to perform its various functionality.
4. Helga support scripts and server configuration - the scaffolding that allows the rest of Helga to function. Should include any tools needed to manage the database or configure the web server, for example.

We expect HelgaWeb, being graphical and easily accessible from anywhere, will be the most heavily used piece of Helga.

We recognize that all people involved in the production of art each have their own unique needs. Therefore, HelgaWeb must be modular, allowing users to easily choose and recombine the specific tools they want to use, creating a personalized workflow.

Each of these four main pieces of Helga will have many facets, and can be further divided into areas of functionality. Helga aims to be as unobtrusive and easy to adopt as possible: therefore, wherever possible these subareas of functionality must not be interdependent.

Assets

The building blocks of a Helga site are assets; your site is a collection of assets. When you create a new project, sequence, or shot, for example, you are creating a new asset. Every asset has "attributes", pieces of data that define

the unique qualities of that asset.

Within Helga each asset is identified by its name. Asset names may only contain letters (upper or lower case), numbers, and underscores. Examples:

- shot1
- uprising
- u2_01
- newShot

Asset names may not begin with "id". "id" is reserved for referring to assets by their unique numerical ID - see "Asset IDs" below.

Asset names do not have to be unique. Since assets are contained in a site-wide pool, it is not reasonable to enforce a rule requiring assets to have names that are unique across the entire site. Asset names can be expected to be similar across projects, such as "shot1" or "renderScript", and Helga does not wish to get in the way of users by limiting the use of these common names.

Asset Types

All assets have an "asset type", which describes what attributes the asset will initially have. Helga ships with a large set of default asset types common to many productions. At the same time, The Helga Project recognizes that each production has different needs. Therefore, it must be possible and quite easy to create new asset types in an implementation of this Specification.

Developer's Note: it is important to note that after an asset is created its asset type is not tied to what attributes it has. The asset type can be changed and attributes may be added or removed; these two activities operate independently of each other.

Asset Lists and Asset Paths

One important asset type that all implementations of this specification must support is the "asset list" type. Asset lists can have "members", other assets that are represented as the list's contents. An asset may be a member of any number of asset lists. As list members may themselves be lists, by creating these list memberships you are able to construct a hierarchy of assets within your site. This hierarchy is commonly referred to as your "asset structure". It provides a means of organizing your site. Assets exist outside of this hierarchy; when an asset is not explicitly defined as the member of any list it still exists in Helga, and by removing an asset from a list at one place in the asset structure you are only removing that list membership.

After a hierarchy of lists has been constructed assets can be represented by a string in the style of a Unix-style file path, e.g. /project1/sequence2/shot2, where the path "/" always represents your Helga site asset. By convention, most project assets should be direct members of the root asset.

Asset lists may also specify a list of asset types that their possible members will be restricted to, i.e. if you specify that the list /users can only have members of the type "user", you cannot add an asset of type "shot" as a member to /users.

An implementation of this specification should allow for "smartlists," asset lists whose memberships are created on the fly by a user-defined series of Helga commands. For example, a smartlist could be created that, when requested for its contents, shows all assets with a certain tag (see "Tags" below). One smartlist that is created by default in Helga is /assets, which shows all assets in Helga. By default this list should only be readable by site administrators.

Developer's Note: The "path" method of referencing assets is quite convenient in most cases, but raises an interesting issue: referencing an asset by path may be ambiguous as there may be more than one asset with that name at that path.

Asset IDs

A partial solution to ambiguous names is that each asset in Helga also has a numerical ID, unique across the entire site. When selecting a member of an asset list, the member assets may be referred to by their numerical IDs. If you wish to refer to an asset by numerical ID, you may use the form "id#", where # is the asset's ID. However, this is often cumbersome as IDs are often kept hidden until explicitly requested; in most cases it is preferable to refer to an asset by its path, so as mentioned it is only a partial solution to name ambiguity.

The Unique Constraint

Another piece of this solution is handled by each asset list individually - asset lists may specify whether the names of their members must be unique. This is known as "the unique constraint". The majority of asset lists in Helga will have this constraint, and should be created with it by default. One example of an asset list that exists by default with no unique constraint will be /assets, which was mentioned above.

It is important to keep in mind whether the asset list you are working in has the unique constraint, and if it does not, to use assets' IDs instead of their names. Properly coded Helga tools should not leave this to the user - they should check for the unique constraint automatically and warn of ambiguous references, or throw an error when given an ambiguous path.

Users may in fact never be aware of this possible ambiguity - in many cases project crew members will be confined by their project managers to access within one asset list that has the unique constraint (see "Permissions").

Current Asset Shorthand

When working within HelgaShell each user has a "current asset path." You may see this referred to as your "current asset" or simply as your "path". Users may change your current asset path at any time. The current asset lets users work faster by allowing the use of shorthand when providing asset paths: instead of constantly needing to provide a full path you may specify only the base name of an asset and the full path will be constructed from a combination of your current path and your input. For example, if your site is organized like this:

- /
 - project1
 - sequence1
 - shot1
 - shot2
 - sequence2
 - shot1
 - shot2

and your current path is /project1, you may refer to the asset /project1/sequence1 as "sequence1", and the asset /project1/sequence1/shot1 as "sequence1/shot1". Upon initializing HelgaShell, the user's current asset is set to whatever the initiating user's "home asset" is. This is generally a project, but can be any asset (a specific shot in your film, for example).

Developer's Note: When referencing an asset, sometimes you need information from its containing list. An asset can have multiple containing lists, but there is never ambiguity as to which list you mean to specify. This is because you either reference an asset by its full path, or use your current asset to build the full path.

Attributes

When a new asset is created, it has a certain set of default attributes in accordance with its type. Additional attributes can then be added or removed from the asset at will.

Developer's Note: Asset types can have parent and child types, forming a tree of type inheritance within your site. When an asset is created it is assigned all attributes from its type and all parent types. Here's an example of what your type inheritance tree might look like, with attributes assigned by that type in quotes. This is not intended to be a complete list:

- *asset (asset ID, name, type ID, description)*
 - *alist (possible member types, sorting method)*
 - *site (helga version #)*
 - *project (full project name, x resolution, y resolution, frames per second)*
 - *shot (start frame, end frame, shot description)*
 - *file (path on disk, creator, date added)*
 - *image (x resolution, y resolution, color space)*
 - *movie*
 - *mayaAscii (maya version #)*
 - *script (platform, version #)*

In this example, if you were to create an asset of type "image", it would inherit attributes from three different places: asset ID, name, type ID, and description from "asset"; path on disk, creator, and date added from "file"; x resolution, y resolution, and color space from "image".

Attribute Inheritance

By combining the concepts of attributes and the hierarchical asset structure, a system of "attribute inheritance" can be created. This inheritance system allows a containing list to provide the value of a requested attribute when the targeted member asset does not have the attribute defined. This system can make its way all the back to the site asset if necessary.

In this way you are able to broadly set an attribute like resolution across a whole project or even your entire site, overriding it only for a handful of assets that are different. To override the value of a project's attribute for a single shot, for example, all you need to do is to add that attribute, with the desired value, to the shot itself.

This inheritance system is extremely versatile. Users can define any attribute they want, assign any value to it, and use the resulting attribute in any they want. The possibilities are endless. Take the example of a rendering script: a Helga-compliant rendering script, given a shot, would first check the shot asset for some values such as x and y resolution. If it doesn't find them, it would defer to perhaps the sequence containing the shot, then the shot's project, then, if necessary, the site itself.

Developer's Note: Because an asset has multiple containing lists and inherits attributes, it sometimes matters what path you refer to an asset at. This asset will always be the same asset, but may inherit different values depending on what containing list it is using. Make sure to keep this in mind, especially when running scripts. This is one of the reasons for the third requirement of Helga scripts: the "test run" (see "Scripts" below). The same is true for permissions (see "Permissions").

Asset Logs

Every asset has a "log" - a chronological record of events in which the asset was involved.

An implementation of this specification should provide an easy standard method of adding log entries, though it is the responsibility of developers to make sure their tools automatically add an entry to an asset's log if it is relevant.

Users with the appropriate access should always be able to add their own custom entries to an asset log. This represents one of the main ideals of Helga: a common standard for peer review on any asset. An example asset log might look like this:

- shot1 was created. - posted by bcipriano - tags: system - 21 Mar 2008 2:45 PM
- board1, type "preview" was added as a member to shot1 - posted by bcipriano - tags: system - 21 Mar 2008 2:46 PM
- board1 was set as the current preview of shot1 - posted by bcipriano - tags: system - 21 Mar 2008 2:47 PM
- reference1, type "file" was added as a member to shot1 - posted by bcipriano - tags: system reference - 21 Mar 2008 2:49 PM
- reference2, type "file" was added as a member to shot1 - posted by bcipriano - tags: system reference - 21 Mar 2008 2:50 PM
- "whoever's working on this shot, check out the two reference files I added for animation reference, or search for log entries with the tag 'reference', I added that tag to both." - posted by bcipriano - tags: user - 21 Mar 2008 2:51 PM

Tasks

Tasks subdivide an asset into ordered bundles of effort that must all be completed for an asset to be considered completed. Some example tasks might be:

- "light shot 12"
- "re-render all shots containing the character 'Operator'"
- "rig the snake"

Tasks must contain the following information:

- a name that follows the same constraints as asset names (letters, numbers, and underscores only, unique within each asset). By default Helga will assign a unique name to new tasks of the form t#.
- a description - a slightly longer block of text that provides more information about the task, such as "light shot 12" - the examples listed above would be descriptions.

Tasks are referenced with the format <asset name>:<task name>. For example, "shot1:t0". You may use current asset shorthand to reference tasks as well - if your current asset was "shot1," you would reference a task like ":t0".

Tasks may have any number of precedents and antecedents which are themselves tasks. You define these connections by creating "links" between tasks. Links may be created between any two tasks within Helga, allowing a chain of tasks dependent on each other to bridge any two assets. In this way a producer can create a chain of tasks for an entire project.

Tasks are associated with exactly one asset. How this association manifests itself is clear for tasks like "light shot 12" and "rig the snake," but perhaps not so clear for a complex task like "re-render all shots containing the character 'Operator'." This is solved in Helga by using "milestone" tasks - tasks that do not describe any amount of additional work, but exist to monitor groups of preceding tasks. For the example task "re-render all shots containing the character 'Operator'," this can be done a few different ways, but all follow generally the same method: begin by creating a new asset list - let's call it "operatorshots" - and adding each shot asset containing the character "Operator" as a member to that list. You may then initiate the task creation process on the asset list, signifying that you wish to apply the task to all member assets. This will create a milestone task on the asset list. This process could differ in a

few ways, e.g. by creating a smartlist instead that automatically queries for all shots with the "operator" tag (see "Tags" below).

This will also create a task on each shot in that list. All of the precedent tasks will have links drawn to the asset list's milestone task automatically, and you will be able to monitor the progress of each of the precedent tasks in greater detail by viewing the milestone task. When all tasks connected to a milestone task are marked as "complete", the status of the milestone task will also be changed to "complete".

Tasks are also associated with one or more users. This allows project managers to assign work to crew members, and for crew members to manage all of their work in one place by simply viewing all tasks associated with them.

Tasks can be "unstarted", "in progress", or "complete". When a task is created it has the default state of "unstarted". Until all of its precedent tasks are marked as "complete", its state can only be changed to "in progress".

An implementation of this specification should allow administrative users to define default task trees for new assets. These default trees should be able to be different for each project and each asset type in each project.

Tags

Assets may be assigned one or more "tags" - short strings of text describing the asset. Tags can be used for querying, site- or project-wide perhaps, for a subset of assets. Tag text may have letters (lowercase or uppercase), numbers, underscores and whitespace. Some example tags:

- protagonist
- effectsShot
- cuttingRoomFloor

By adding the "protagonist" tag to all of the shots in your film that contain your protagonist character, you would be able to easily get a list all of those shots, and tools could be made to take advantage of this on-the-fly list.

Permissions

Permissions are associated with exactly one asset and exactly one user. There are four levels of permissions in Helga:

- "none" - you are allowed no access to this asset - you may not read or write any data from it.
- "basic" - you are only allowed read access. You may:
 - View the asset.
 - Read the asset's attribute values and tags.
 - View the asset's log.
- "advanced" - you are allowed read and limited write access. In addition to guest permissions, you may:
 - Create new assets (which you then have "admin" access on).
 - View tasks, links, and task attributes.
 - Add new tags to assets, tasks, and log entries.
 - Add entries to the asset's log.
 - Create new attributes.
 - Modify existing attributes.
 - Execute scripts.
- "admin" - you have full read and write access. In addition to guest and crew permissions, you may:
 - Remove attributes.
 - Remove list memberships.
 - Modify tasks.
 - Add new assets.

Permissions on asset lists are inherited by their members, and an asset's inherited permission level is overridden by explicitly defining a different permission level on the asset itself. For example, let's assume you have an asset structure like this:

- /
 - project1
 - project2
 - scripts
 - users

If you have admin permissions on the top-level asset. "/", then you have admin permissions on every asset in Helga - project1, project2, scripts, users, and all members of those lists. This effectively makes you a site administrator.

By default every Helga site has a "guest" user, who also has "basic" permissions on /, giving them basic permissions on every asset in Helga. Like everything else in Helga this is not set in stone: the guest user's access to assets can be restricted in several ways:

- Their permission level on / could be set to none, and set to "basic" only on a per-asset basis, for example on only one or two projects.
- Only those projects that wish to deny access to the guest user could explicitly set the guest user's permission to "none" on their project root, blocking the guest user from any access.
- The guest user could be removed entirely, allowing only registered users access to that particular Helga site.

Developer's Note: This has been implied, but is important enough to outline specifically: there is a difference between having a permission level of "none" and having no permission set explicitly. With no permission level set, Helga will defer to the permission level set on the containing list asset, walking all the way back up to / if necessary. If Helga still finds no explicitly set permission, it will default to "none". However, if Helga in its initial check finds a permission level explicitly set to "none" it will not defer. This allows administrators to explicitly block access to certain assets.

Users

Users are assets in Helga. "user" is one of the default asset types in Helga.

When adding a user, you do not need to choose a path for it - whatever path you provide, it will be added as a member of the /users asset list. When a user is created they have guest permissions on /.

Scripts

"Script" has a fairly loose definition in Helga: a script is a file on disk, which will perform some sort of automated task. What the script does may vary greatly - a script could fix reference paths on a 3d scene file, a script could composite two images, or a script could spool a render to your local distribution system such as DrQueue or Pixar's Alfred, to name a few common uses. Using the Helga script system you can easily control and automate any scriptable program on your Helga server or, if you have a local distribution system, on any of the computers in your local network.

The requirements for writing Helga compliant scripts are few, and there are several benefits:

- The developer gets a web interface auto-built for his or her script, fully integrated into the Web Interface.
- Power users are able to use the script through the Helga Shell, a system they are already familiar with.

- The developer saves time by having to write less code: several things are done automatically, including asset type checking and user validation (see below).

Asset Placement

Scripts are assets in Helga. It is a minority of situations for which a script will be written for a single asset. In these cases the convention for placement within Helga is apparent - you could simply add the script as a member of the asset which it will operate on. For many other cases it is necessary for the script to exist on a per-project or even a per-site basis. For a script to be available to an asset and all of its member assets, the convention is to create a new asset list at <the asset>/scripts, and place your scripts in that asset list. It should be noted that most scripts will not rely on their placement within an asset structure; they will be passed an asset or a file to operate on each time they are used.

Type Control and User Validation

You may control which members a script may operate on by restricting its possible target asset to one or more asset types. The script will only be able to accept as input an asset of the given type. In this way we are able to write a script that, for example, composites two images together using Apple's Shake. Shake only accepts .shk files, so we might create a new asset type called "shakeFile", and define our script as operating only on that asset type.

Normal user permissions apply to scripts - if a user only has guest access on /project1, for example, they will not be able to execute a script stored under /project1/scripts, unless they have at least "crew" access on either /project1/scripts or on the script itself.

Helga Compliance

There are a few things that every Helga compliant script should be able to do:

- Return usage information - the arguments that need to be supplied to the script for it to run and how those values affect the outcome of the script.
- Return interface building information - how each argument could be supplied in a visual interface - should it be a textbox, a pull down menu, a checkbox?
- Do a "test run" - build all of its argument data and display information about what would have happened if the script had executed. Scripts can pull their input data from many different places, so it is important for users to be able to see what data the script is planning on using without running the script itself.

How to satisfy these requirements within implementations of this Specification will differ; please consult your implementation specific scripting guidelines for more information. If you are creating an implementation of this Specification, make sure to very clearly lay out how to write Helga-compliant scripts within your implementation.

Machines

One more type of asset in Helga is the machine. A machine asset represents a computer related to your Helga site. The machine's relationship to Helga could be of many sorts. You might have a separate computer as your fileserver, or as the head node of some distribution system. By adding some information about that computer as an asset in Helga, you allow the integration of these different machines with the rest of your Helga set up.

Like users, machines override their given location in the asset structure. They can only be created under the asset list /machines.

File Versioning

Under discussion. Should be covered in version 1.1 of this Specification.

The Publish/Subscribe System

Under discussion. Should be covered in version 1.1 of this Specification.