

GenDL Unified Documentation

Dave Cooper

October 11, 2012

Contents

1	Introduction	1
1.1	Knowledge Base Concepts According to Genworks	1
1.2	Goals for this Tutorial	1
1.3	What is GenDL?	2
1.4	Why GDL (what is GDL good for?)	2
1.5	What GDL is not	3
2	Installation	5
2.1	Installation of pre-packaged Gendl	5
2.1.1	Download the Software and retrieve a license key	5
2.1.2	Unpack the Distribution	5
2.1.3	Test your Installation	6
2.1.4	Make a Desktop Shortcut	7
2.1.5	Populate your Initialization File	7
2.2	Installation of open-source Gendl	7
2.2.1	Install and Configure your Common Lisp environment	7

Chapter 1

Introduction

1.1 Knowledge Base Concepts According to Genworks

You may have an idea about Knowledge Base Systems, or Knowledge *Based* Systems, from college textbooks or corporate marketing propaganda, and found the concept too broad to be of practical use. Or you may have heard jabs at the pretentious-sounding name, “Knowledge-based Engineering,” as in: “you mean as opposed to Ignorance-based Engineering?”

To provide a clearer picture, we hope you will agree that our concept of a KB system is simple and practical, and in this tutorial our goal is to make you comfortable and excited about the ideas we have implemented in our flagship system, GenDL (or “Gendl”)

Our definition of a *Knowledge Base System* is an object-oriented programming environment which implements the features of *Caching* and *Dependency tracking*. Caching means that once the KB has computed something, it might not need to repeat that computation if the same question is asked again. Dependency tracking is the flip side of that coin — it ensures that if a cached result is *stale*, the result will be recomputed the next time it is *demand*ed, so as to give a fresh result.

1.2 Goals for this Tutorial

This manual is designed as a companion to a live two-hour GDL/GWL tutorial, but you may also be reading it on your own. In either case, the basic goals are:

- Get you excited about using GDL/GWL
- Enable you to judge whether GDL/GWL is an appropriate tool for a given job
- Arm you with the ability to argue the case for using GDL/GWL when appropriate
- Prepare you to begin maintaining and authoring GDL/GWL applications, or porting apps from similar KB systems into GDL/GWL.

This manual will begin with an introduction to the Common Lisp programming language. If you are new to Common Lisp: congratulations! You have just discovered a powerful tool backed by a powerful standard specification, which will protect your development investment for decades to come. In addition to the brief overview in this manual, many resources are available to get you

started in CL — for starters, we recommend Basic Lisp Techniques¹, which was prepared by the author of this tutorial.

1.3 What is GenDL?

GenDL (or Gendl to be a bit more relaxed) is an acronym for “General-purpose Declarative Language.”

GenDL is a superset of ANSI Common Lisp, and consists mainly of automatic code-expanding extensions to Common Lisp implemented in the form of macros. When you write, let’s say, 20 lines in GenDL, you might be writing the equivalent of 200 lines of Common Lisp. Of course, since GenDL is a superset of Common Lisp, you still have the full power of the CL language at your fingertips whenever you are working in GenDL.

Since GDL expands into CL, everything you write in GDL will be compiled “down to the metal” to machine code with all the optimizations and safety that the tested-and-true CL compiler provides. This is an important distinction as contrasted to some other so-called KB systems on the market, which are really nothing more than interpreted scripting languages which often impose arbitrary limits on the size and complexity of your application.

GenDL is also a true *declarative* language. When you put together a GDL application, you write and think mainly in terms of objects and their properties, and how they depend on one another in a direct sense. You do not have to track in your mind explicitly how one object or property will “call” another object or property, in what order this will happen, etc. Those details are taken care of for you automatically by the language.

Because GDL is object-oriented, you have all the features you would normally expect from an object-oriented language, such as

- Separation between the *definition* of an object and an *instance* of an object
- High levels of data abstraction
- The ability for one object to “inherit” from others
- The ability to “use” an object without concern for its “under-the-hood” implementation

GDL supports the “message-passing” paradigm of object orientation, with some extensions. Since full-blown ANSI CLOS (Common Lisp Object System) is always available as well, the Generic Function paradigm is supported as well. Do not be concerned at this point if you are not fully aware of the differences between these two paradigms².

1.4 Why GDL (what is GDL good for?)

- Organizing and interrelating large amounts of information in ways not possible or not practical using conventional languages or conventional relational database technology alone;

¹ BLT is available at http://www.franz.com/resources/educational_resources/cooper.book.pdf

²See Paul Graham’s ANSI Common Lisp, page 192, for an excellent discussion of the Two Models of Object-oriented Programming.

- Evaluating many design or engineering alternatives and performing various kinds of optimizations within specified design spaces;
- Capturing the procedures and rules used to solve repetitive tasks in engineering and other fields;
- Applying rules to achieve intermediate and final outputs, which may include virtual models of wireframe, surface, and solid geometric objects.

1.5 What GDL is not

- A CAD system (although it may operate on and/or generate geometric entities);
- A drawing program (although it may operate on and/or generate geometric entities);
- An Artificial Intelligence system (although it is an excellent environment for developing capabilities which could be considered as such);
- An Expert System Shell (although one could be easily embedded within it).

Without further ado, then, let's turn the page and get started with some hands-on GDL...

Chapter 2

Installation

This section will take you through the installation of Gendl from a prepackaged distribution with the Allegro CL Common Lisp engine and the Slime IDE (based on Gnu Emacs). Gendl is also available as open-source software; if you want to use that version, then you are expected to have your own Common Lisp engine installed and set up, and your own editing environment. See the README file with the open-source distribution¹ for more details on using it.

2.1 Installation of pre-packaged Gendl

2.1.1 Download the Software and retrieve a license key

1. Visit the Downloads section of the [Genworks Newsite](#)
2. Enter your email address².
3. Download the latest Payload and gpl.zip for Windows.
4. Click to receive license key file by email.

2.1.2 Unpack the Distribution

GenDL is currently distributed for all the platforms as a self-contained “zip” file which does not require official administrator installation. What follows are general instructions; more up-to-date details may be found in the email which accompanies the license key file. A five-minute installation video is also available in the Documentation section of the [hrefhttp://genworks.com/newsite](http://genworks.com/newsite)Genworks Newsite.

1. Unzip the gdl1581... zipfile into a location where you have write permissions
2. Unzip the gpl.zip file at the same level as the gdl payload
3. Copy the license key file as gdl.lic (for Trial, Student, Professional editions), or devel.lic (for Enterprise edition) into the `program/` directory within the gdl1581.../ directory.

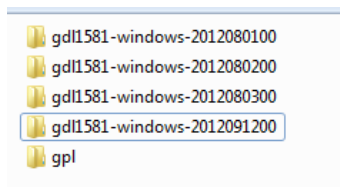


Figure 2.1: Several Gendl versions and one GPL

So you now should have two directories at the same level: one named `gdl1581...` (the rest of the name will contain the specific dated build stamp), and a `gdl/` directory at the same level. Note that as seen in Figure 2.1, it is possible to have several Gendl versions installed, but just a single common `gpl/` folder.

2.1.3 Test your Installation

Test your installation according to the following checklist:

1. Invoke the `run-gdl.bat` (Windows), or `run-gdl` (Linux, MacOS) startup script. This should launch Gnu Emacs with a README file displayed by default. Take the time to look through this README file. Especially the later part of the file contains information about Emacs keyboard shortcuts available.
2. In emacs, enter: `M-x glime`. That is, hold down the “Meta” (or “Alt”) key, and press the “M” key. You will see this command shown in the *mini-buffer* at the bottom of the Emacs window, as shown in Figure ??
3. press the “Enter” key

4. On Windows, you will get a new window, named the Allegro CL Console or the Gendl Console. Watch this console for any errors or warnings.

The first time you start up, you may see messages in this console for several minutes while the system is being built (or if you received a completely pre-built system, the messages may last only a few seconds).

On Linux or MacOS, there will be a separate Emacs buffer (available through Emacs’ “Buffers” menu) where you will see these messages.

The messages will consist of compiling and loading information, followed by copyright and welcome information for Gendl. After these messages have finished, you should see the following command prompt: `gdl-user(1):`

5. In your Web Browser (e.g. Google Chrome, Firefox, Safari, Opera, Internet Explorer), perform the following steps:

(a) visit <http://localhost:9000/tasty>.

¹<http://github.com/genworks/Genworks-GDL/>

²if your address is not on file, send mail to licensing@genworks.com

- (b) accept default robot:assembly.
- (c) Select “Add Leaves” from the Tree menu.
- (d) Click on the top node in the tree.
- (e) Observe the wireframe graphics for the robot.
- (f) Click on the robot to zoom in.
- (g) Select “Clear View!” from the View menu.
- (h) Select “X3DOM” from the View menu.
- (i) Click on the top node in the tree.
- (j) “Refresh” or “Reload” your browser window (may not be necessary).
- (k) If your browser supports WebGL, you will see the robot in shaded dynamic view.
- (l) Select “PNG” from the View menu.
- (m) You will see the wireframe view of the robot as a PNG image.

2.1.4 Make a Desktop Shortcut

2.1.5 Populate your Initialization File

The default initialization file for Gendl is called `gdlnit.cl`,

2.2 Installation of open-source Gendl

2.2.1 Install and Configure your Common Lisp environment

Gendl is currently tested to build on the following Common Lisp engines:

- Allegro CL (commercial product from Franz, free Express Edition available)
- LispWorks (commercial product from LispWorks Ltd, free Personal Edition available)
- Steel Bank Common Lisp (SBCL) (free open-source project with permissive license)

Please refer to the documentation for each of these systems for full information on installing and configuring the environment. Typically this will include a text editor, either Gnu Emacs with Superior Lisp Interaction Mode for Emacs (Slime), or a built-in text editing and development environment which comes with the Common Lisp system.

Index

Basic Lisp Techniques, 2

Caching, 1

Common Lisp, 1

compiled language
 benefits of, 2

declarative, 2

Dependency tracking, 1

Ignorance-based Engineering, 1

Knowledge Base System, 1

macros

 code-expanding, 2

object-orientation

 generic-function, 2

 message-passing, 2