GenDL Unified Documentation

Dave Cooper

October 15, 2012

Contents

Chapter 1

Introduction

1.1 Welcome

Congratulations on your purchase or download of Genworks Gendl. By investing some of your valuable time into learning this system, you are investing in your future productivity and you are becoming part of a quiet revolution. Although you may have come to Genworks Gendl because of an interest in 3D modeling or mechanical engineering, you will find that a whole new world, and a whole new approach to computing, will now be at your fingertips.

1.2 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 *demanded*, so as to give a fresh result.

1.3 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 Techniques[1], which was prepared by the author of this tutorial.

1.4 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 propery, 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

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

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 paradigms[2].

1.5 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;
- 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.

1.6 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...

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

Chapter 2

Installation

Follow Section 2.1 if your email address is registered with Genworks and you will install a pre-packaged Gendl distribution including its own Common Lisp engine. Gendl is also available as open-source software[1]; if you want to use that version, then please refer to Section 2.2.

2.1 Installation of pre-packaged Gendl

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).

2.1.1 Download the Software and retrieve a license key

1. Visit the Downloads section of the Genworks Newsite
2. Enter your email address[2].
3. Download the latest Payload and gpl.zip for Windows[3]
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 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.

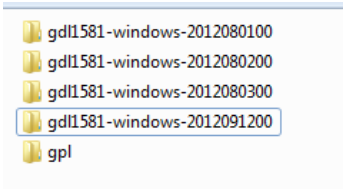

Figure 2.1: Several Gendl versions and one GPL

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 Make a Desktop Shortcut

1. Using the "My Computer" or "Computer" Windows file manager, right-mouse on the `run-gdl.bat` file.
2. Select "Create Shortcut."
3. Now drag the new "Run-gdl-shortcut" icon to your desktop.

2.1.4 Populate your Initialization File

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

2.2 Installation of open-source Gendl

This section is only relevant if you have not received a pre-packaged Gendl distribution with its own Common Lisp engine. If you have received a pre-packaged Gendl distribution, then please skip this section. In case you want to use the open-source Gendl, you will use your own Common Lisp installation and fetch Gendl (Genworks-GDL) using a very powerful and convenient CL package/library manager called *Quicklisp*.

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 Inc, 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)

[1]http://github.com/genworks/Genworks-GDL
[2]if your address is not on file, send mail to licensing@genworks.com
[3]If you already have a gpl.zip from a previous Gendl installation, it is not necessary to download a new one.

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.

As of this writing, a convenient way to set up Emacs with Slime is to use the Quicklisp-slime-helper.

2.2.2 Load and Configure Quicklisp

As of this writing, Quicklisp is rapidly becoming the defacto standard library manager for Common Lisp.

- Visit the Quicklisp website
- Follow the instructions there to download the `quicklisp.lisp` bootstrap file and load it to set up your Quicklisp environment.

2.3 System Startup and Testing

2.3.1 System Startup

Startup of prepackaged Gendl distribution

To start a prepackaged system, follow these steps:

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 "X" key, then type "glime." You will see this command shown in the *mini-buffer* at the bottom of the Emacs window, as shown in Figure 2.2

3. press the "Enter" key

4. On Windows, you will get a new window, named the Allegro CL Console or the Gendl Console, as shown in Figure 2.3. This window might start out in minimized form (as an icon at the bottom of your screen). Click on it to open it. 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:

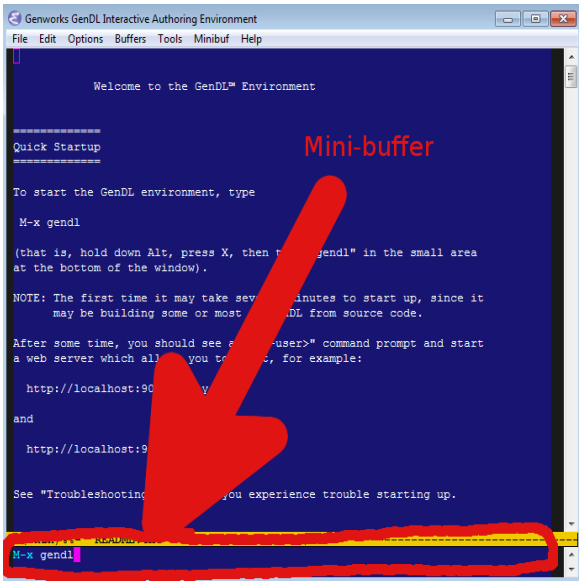

Figure 2.2: The mini-buffer in Emacs

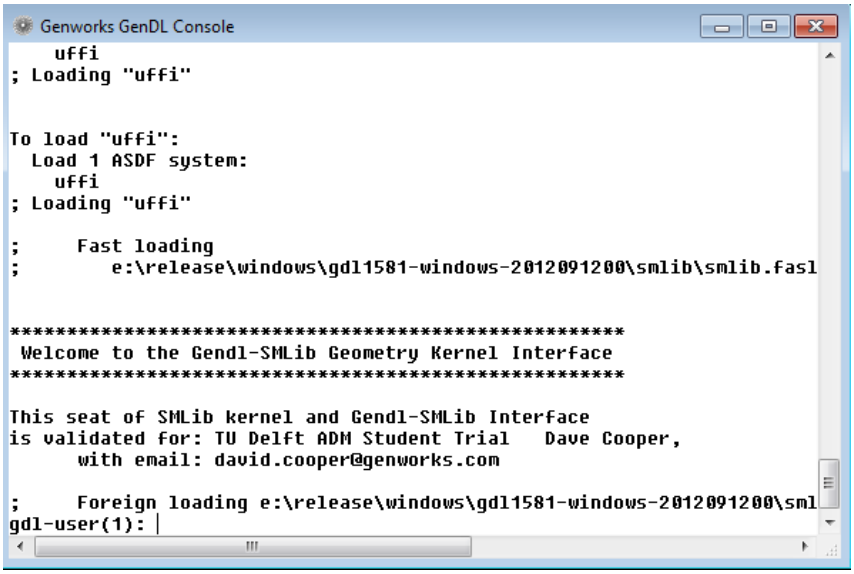

Figure 2.3: Genworks Gendl Console

`gdl-user(1):`

The Genworks GDL console contains a command prompt, but mostly you will use the Slime REPL buffer in Emacs to type commands. The Genworks GDL console is mainly used for displaying output text from the Gendl system and from your application.

Startup of open-source Gendl distribution

To start an Open-source distribution, follow these steps:

1. Start your Common Lisp engine and development environment (e.g. SBCL with Emacs and Superior Lisp Interaction Mode for Emacs).
2. After Quicklisp is installed and initialized in your system, type: `(ql:quickload :genworks-gdl)` to get Genworks Gendl installed and loaded in your environment.
3. Type the following to initialize the Gendl environment:

   ```
   (gdl:start-gdl :edition :open-source)
   ```

2.3.2 Basic System Test

You may test your installation using the following checklist. These tests are optional. You may perform some or all of them in order to ensure that your Gendl is installed correctly and running smoothly. In your Web Browser (e.g. Google Chrome, Firefox, Safari, Opera, Internet Explorer), perform the following steps:

1. visit http://localhost:9000/tasty.
2. accept default robot:assembly.
3. Select "Add Leaves" from the Tree menu.
4. Click on the top node in the tree.
5. Observe the wireframe graphics for the robot as shown in 2.4.
6. Click on the robot to zoom in.
7. Select "Clear View!" from the View menu.
8. Select "X3DOM" from the View menu.
9. Click on the top node in the tree.
10. "Refresh" or "Reload" your browser window (may not be necessary).
11. If your browser supports WebGL, you will see the robot in shaded dynamic view as shown in Figure 2.5.
12. Select "PNG" from the View menu. You will see the wireframe view of the robot as a PNG image.
13. Select "X3D" from the View menu. If your browser has an X3D plugin installed (e.g. BS Contact), you will see the robot in a shaded dynamic view.

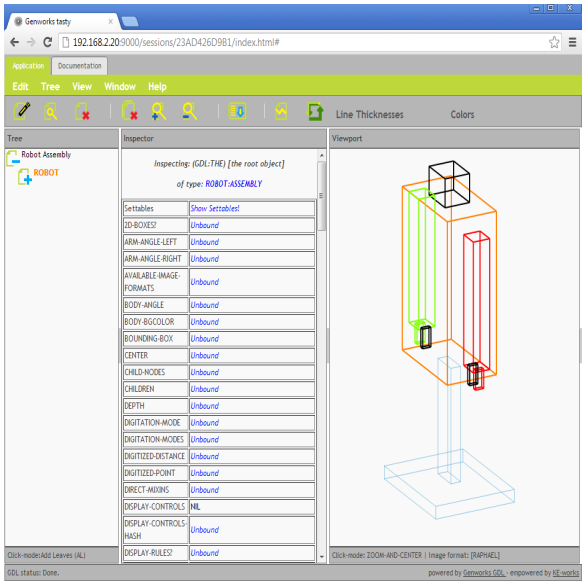

Figure 2.4: Robot displayed in Tasty

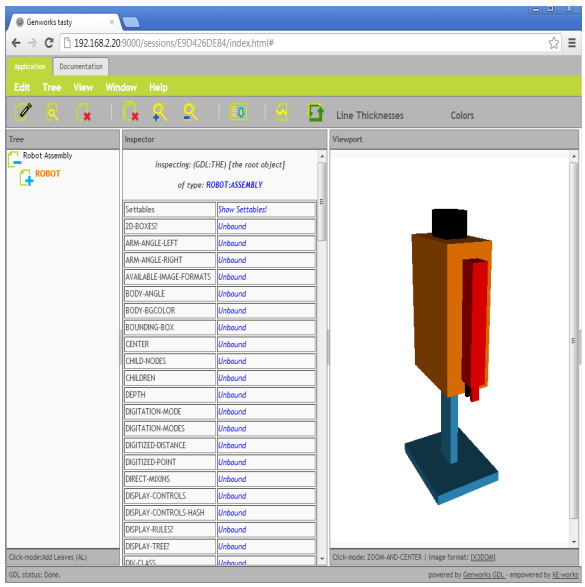

Figure 2.5: Robot x3dom

2.3.3 Full Regression Test

The following commands will invoke a full regression test, including a test of the Surface and Solids primitives provided by the SMLib geometry kernel. Note that the SMLib geometry kernel is only available with proprietary Gendl licenses — therefore if you have an open-source or Trial version, you these regression tests will not all function.

In Emacs at the Slime REPL prompt, type the following commands:

1. `(ql:quickload :gdl-regression)`
2. `(gdl-lift-utils::define-regression-tests)`
3. `(gdl-lift-utils::run-regression-tests-pass-fail)`
4. `(pprint gdl-lift-utils::*regression-test-report*)`

2.4 Getting Help and Support

If you run into unexplained errors in the installation and startup process, please contact the following resources:

1. Make a posting to the Genworks Google Group
2. For pure Common Lisp issues, join the #lisp IRC (Internet Relay Chat) channel and discuss issues there.
3. Also for Common Lisp issues, follow the comp.lang.lisp Usenet group.
4. If you are a supported Genworks customer, send email to support@genworks.com
5. If you are not a supported Genworks customer but you want to report an apparent bug or have other suggestions or inquiries, you may also send email to support@genworks.com, but please understand that Genworks cannot guarantee any response or a particular timeframe for any response.

Chapter 3

Basic Operation of the Gendl Environment

Gendl is a dynamic language environment with incremental compiling and in-memory definitions. That means that as long as the system is running, you can *compile* new *definitions* of functions, objects, etc, and they will immediately become available as part of the running system. If you shut down and restart the system, then you will have to recompile and/or reload those definitions in order to bring the system back into the same state. This is typically done automatically, using commands placed into the `gdlinit.cl` initialization file.

3.1 Startup and Shutdown

The typical Gendl environment consists of two programs: Gnu Emacs (the editor), and Common Lisp engine (e.g. the `mlisp.exe` executable of Allegro CL). Emacs runs as the main *process*, and this in turn starts the Common Lisp engine as a *sub-process*.

As introduced in Chapter 2, the typical way to start a pre-packaged Gendl environment is with the `run-gdl.bat` (Windows), or `run-gdl` (MacOS, Linux) script files. Invoke this script file from your computer's file manager, from the command-line, or from a desktop shortcut if you have created one as outlined in section 2.1.3.

Index