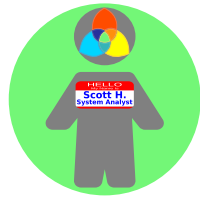


Logical Map How-to Guide



Scott H., System Analyst











2025-03-05

Overview

As we face quickly changing systems, knowing how to create a logical map is critical. There are many existing tools that help with this. I distill down to the quickest and most effective methods, combining material and data flow with just three symbol classes and two types.

Introduction

A logical map makes sense of related concepts by limiting the visualization to necessary functional elements. Triple System Analysis () and Adaptive Analysis () explain how to use multi-level maps to understand systems ([H. 2023](#)  ([H. 2024](#) ). This how-to guide () builds on those ideas, but is limits the domain to materials and data, which are core aspects of concern. Data might live on paper, a computer hard disk, as memories of childhood, or shared cultural forms in our dreams ([Samson et al. 2023](#))  . Data flows in our daily lives as we run reports, write in our diary, account for money, message our friends, maintain contacts, and many other things that rely on other services to process and store our data. Those services rely on others. Restaurants often require data flow to accept customers, as orders and payment are handled by external services. When there is a network interruption, business grinds to a halt. Supply chains of production and distribution of materials like eggs, steel, mobile phones or lettuce can also cause significant disruption when interrupted. Creating a map of these flows prior to failure can help recovery. Often it is cost-prohibitive to map all possible flows prior to failure; however, the methods describe by this paper our intended to be quick enough to use at time of failure. Data flow maps were introduced in the 1970s to analyze complicated data processing, and were proven to be an effective cognitive aid ([Gane and Sarson 1977](#))  . Fig. 1 is an example map that uses adopts the symbols and conventions

The rounded rectangle blue nodes are transformations. The teal rectangle nodes are agents that are the sources or sinks of data or intelligence. The reddish-brown nodes store data or materials at rest. Dotted lines within the node represent data. Solid lines represent materials. As I explained in  , data flow diagrams are behind agents that operate transforms. This is why I think it is OK to mix the nodes, as most of the function is behind the screens, the black box of the device or report that assists the transform. Magenta dots in the corner of a transform/process node mean the node can be expanded by clicking. An orange dot means that notes and narrative will show with a pointer hover. A blue dot in the lower right corner of a material transform means there is a connection to the associated full data flow.

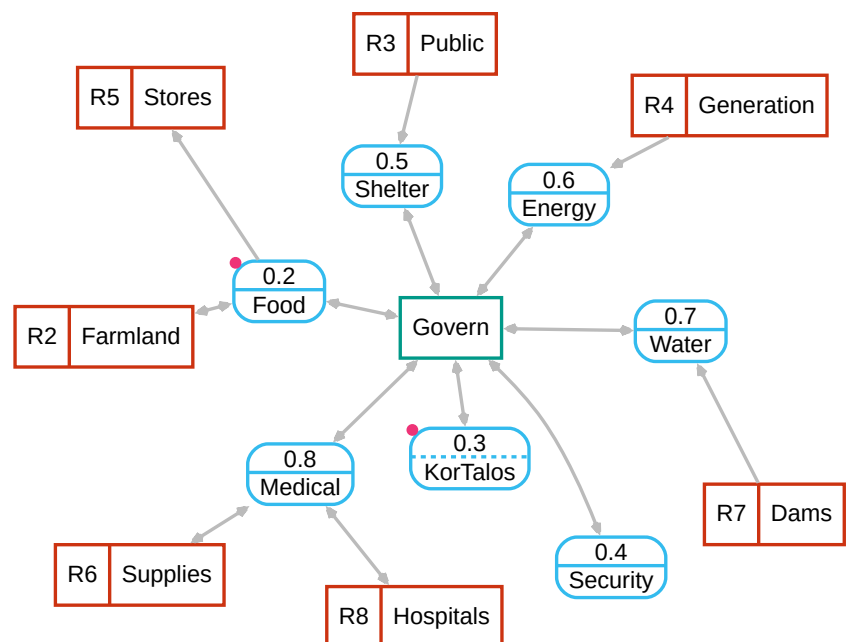





Figure 1: Top

Creating a Map

 introduced the graph stack format. Let's use that to create the graph in Fig. 1. We need to install software that will convert the text model to a standard graph format. I'm going to assume that you can use NPM, or, at least can bribe a programmer with a beverage and food to help you ("[Node.js — an Introduction to the Npm Package Manager](#)" n.d.) . We will continue to build out more sophisticated features, but for now, create a blank directory, cd into it, and install text-model-dot, gsdot-svg, and http-server with NPM:

```
1 $ npm install text-model-dot gsdot-svg http-server
2 added 101 packages in 3s
```

We now have a new directory called node_modules which contains the software. It is also possible to go directly to the repository on GitHub and download the software, but getting used to NPM will help us later on ([H. 2025](#)) . Create a file named index.html in the directory with this content:

```
1 <!DOCTYPE html><html lang="en"><title>Top</title>
2 <div id="map"></div>
3 <script type="module">
4   import { model_to_dots } from "./node_modules/text-model-dot/text-model-dot.js"
5   import { gsdot_svg } from "./node_modules/gsdot-svg/dist/gsdot-svg.bundle.js"
6   async function main() {
7     const model = await (await fetch('./node_modules/text-model-dot/example.txt')).text()
8     const dot_lines = model_to_dots(model).dots["Top"];
9     await gsdot_svg(dot_lines, 'default', 'map');
10  }
11  main()
12 </script></html>
```

Start up a web server to see the graph:

```
1 $ npx http-server
2 Starting up http-server, serving ./
3
4 http-server version: 14.1.1
5
6 http-server settings:
7 CORS: disabled
8 Cache: 3600 seconds
9 Connection Timeout: 120 seconds
10 Directory Listings: visible
11 AutoIndex: visible
12 Serve GZIP Files: false
13 Serve Brotli Files: false
14 Default File Extension: none
15
16 Available on:
17   http://127.0.0.1:8080
18 Hit CTRL-C to stop the server
```

Browse to <http://127.0.0.1:8080>, and you'll see a graph like Fig. 1.

References

- Gane, Chris, and Trish Sarson. 1977. *Structured Systems Analysis: Tools and Techniques*. McDonnell Douglas Systems Integration Company. <https://archive.org/details/structuredsystem0000gane>.
- H., Scott. 2023. "Triple System Analysis," May. <https://doi.org/10.5281/ZENODO.7826793>.
- . 2024. "Adaptive Analysis," August. <https://doi.org/10.5281/ZENODO.13684896>.
- . 2025. "Acodrst/Text-Model-Dot." <https://github.com/acodrst/text-model-dot>.
- "Node.js — an Introduction to the Npm Package Manager." n.d. Accessed March 2, 2025. <https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager>.
- Samson, David R., Alice Clerget, Noor Abbas, Jeffrey Senese, Mallika S. Sarma, Sheina Lew-Levy, Ibrahim A. Mabulla, et al. 2023. "Evidence for an Emotional Adaptive Function of Dreams: A Cross-Cultural Study." *Scientific Reports* 13 (1): 16530. <https://doi.org/10.1038/s41598-023-43319-z>.

