LegalSS

Release 0.9

Meng Weng Wong

Jan 11, 2023

CONTENTS:

CHAPTER

ONE

WHAT IS L4 GOOD FOR?

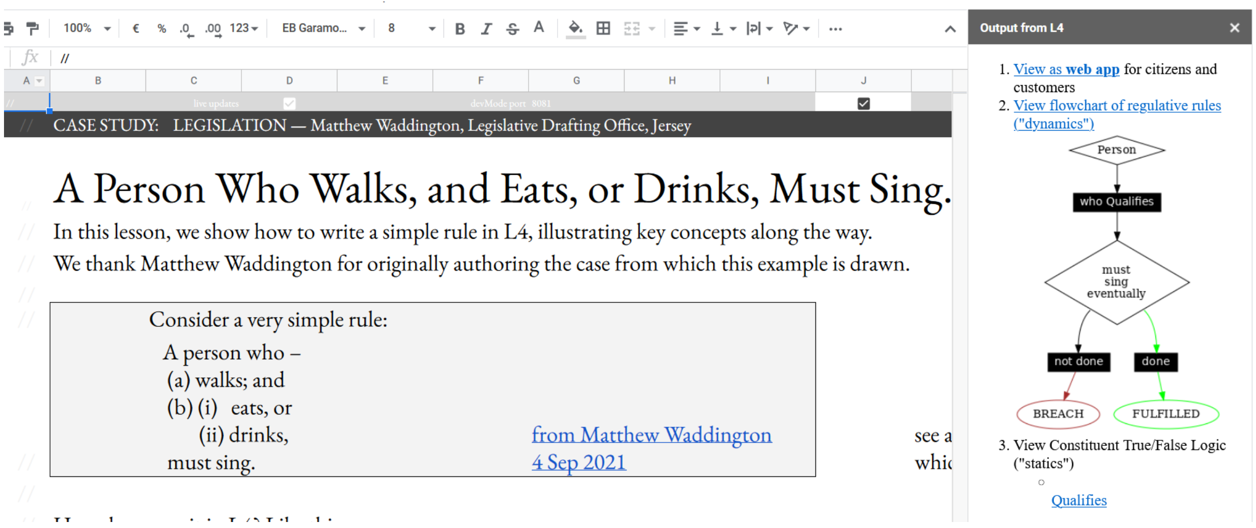
L4 offers a low-code way for non-lawyers to explore existing contracts and legislation, and to generate new contracts and regulations.

If the contract you need is already available in the L4 package library, you can just fill in the blanks and generate a document that you can sign, after, of course, running it past your lawyers for review.

The L4 package library will soon contain loan agreements, leasing agreements, and investment agreements like the SAFE. It also contains encodings of insurance policies based on those offered by major insurers, so you can better understand what your medical or travel insurance actually covers.

If you want to customize an existing contract template, you can fork and edit it in the LegalSS Google Sheets. If you click on the + buttons in the left margin of the spreadsheet, you should see the expanded content.

To make life easier for "legal engineers", L4 generates convenient visualizations of the logic and the moving parts of your "legal program".



CASE STUDY: LEGISLATION — Matthew Waddington, Legislative Drafting Office, Jersey

A Person Who Walks, and Eats, or Drinks, Must Sing.

In this lesson, we show how to write a simple rule in L4, illustrating key concepts along the way.

We thank Matthew Waddington for originally authoring the case from which this example is drawn.

Consider a very simple rule:

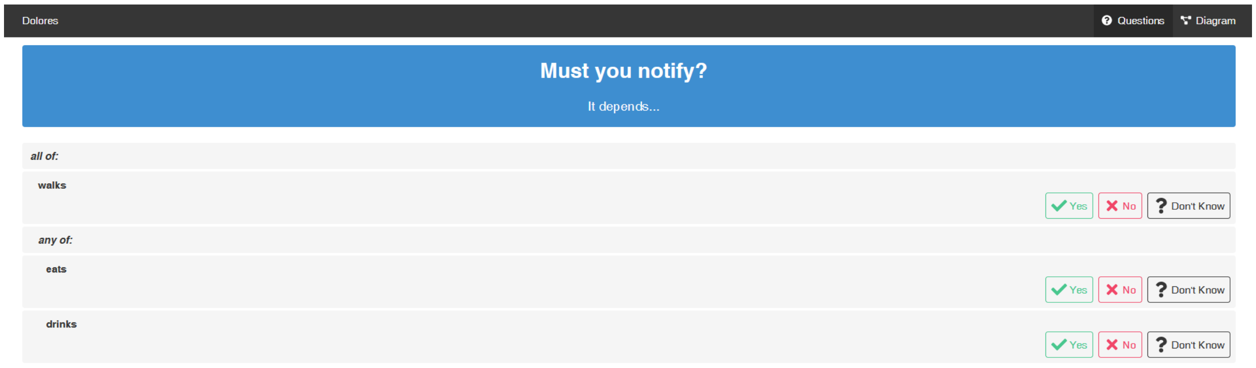
A person who –
(a) walks; and
(b) (i) eats, or
(ii) drinks,
must sing.

from Matthew Waddington
4 Sep 2021

1. View as **web app** for citizens and customers
2. View flowchart of regulative rules ("dynamics")
3. View Constituent True/False Logic ("statics")
   - Qualifies

L4 also sanity-checks your programs to detect internal conflicts and loopholes.

L4 automatically generates a web app that helps end users explore the logic.



In addition to contracts, L4's package library also contains encodings of legislation and regulation in the areas of data privacy and building permission.

The entire "Quickstart" tab is a tutorial. The other tabs in the LegalSS spreadsheet show examples of L4 at work, and illustrate common real-world usage in both contractual and legislative environments.

1.1 Documentation Timeline

This documentation was written in Oct/Nov 2022. The system is still rough around the edges.

In the future there will be more helpful documentation in the FAQ, depending on the kinds of problems that tend to occur.

CHAPTER

TWO

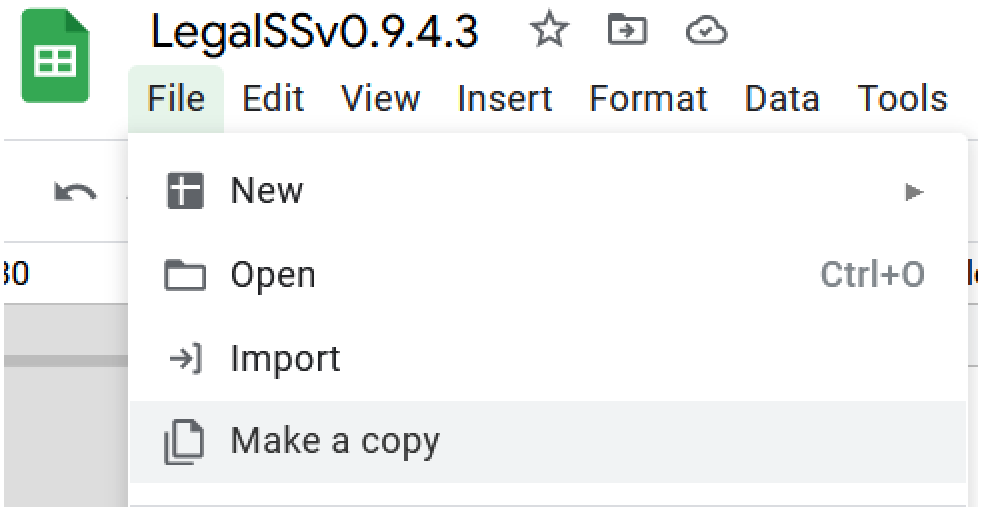
SETTING UP THE LEGALSS GOOGLE SHEET

Setting up the LegalSS spreadsheet introduces you to the output sidebar and lets you explore Google Sheets as an Interactive Development Environment (IDE).

Clone the spreadsheet so you can try the exercises yourself.

2.1 How to clone the spreadsheet

1. Clone the spreadsheet by clicking on 'File' at the toolbar, then choose the 4th option 'Make A Copy'.

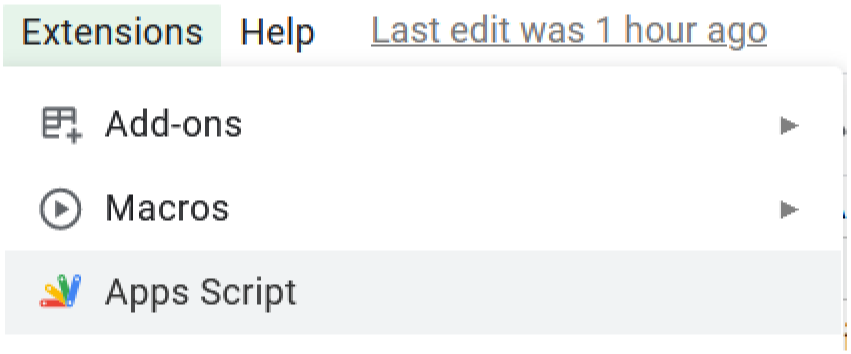


2. Rename the result: this will be your copy of the spreadsheet tutorial.

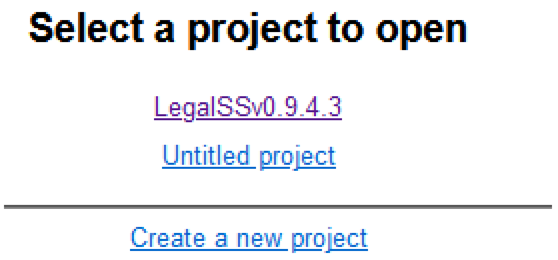
2.2 Activating the L4 Sheets IDE

A one-time procedure is needed to activate the L4 Sheets IDE.

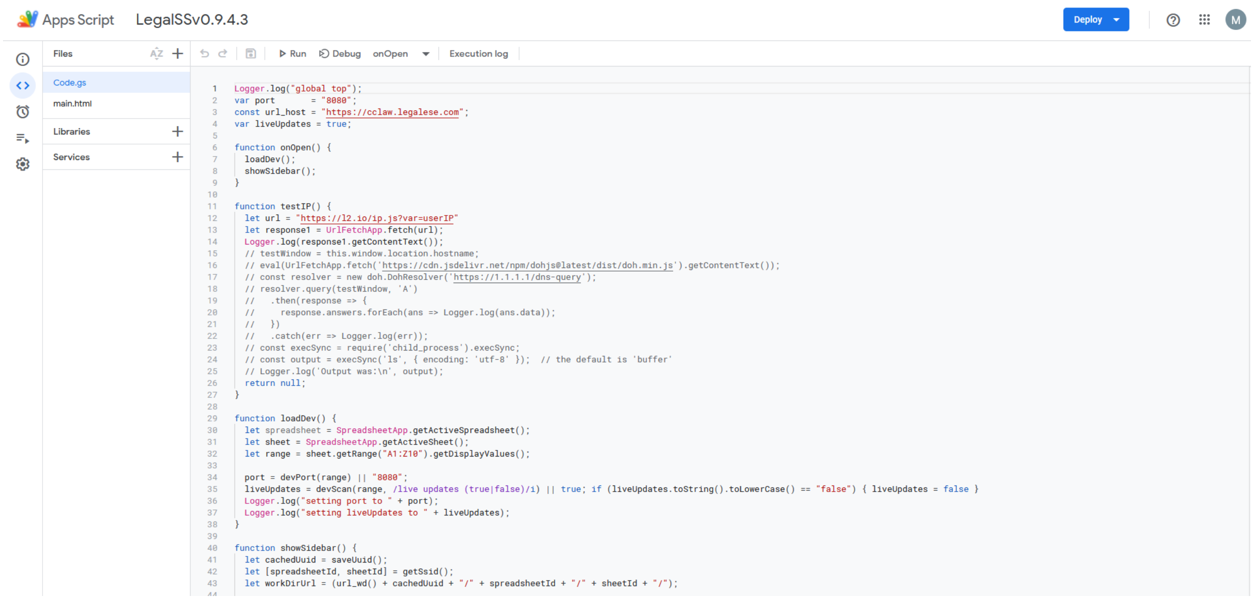
1. Click on Extensions/Apps Script.



2. A new tab will open on your browser and you will be asked to select a project to open. Choose the first project "LegalSS…" with the version number (e.g. "LegalSSv0.9.4.3"). Do not click on "Untitled Project".

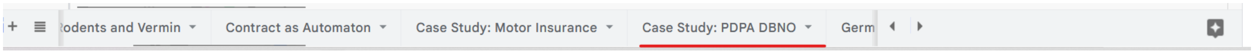


You should be directed to a page similar to the below screenshot.

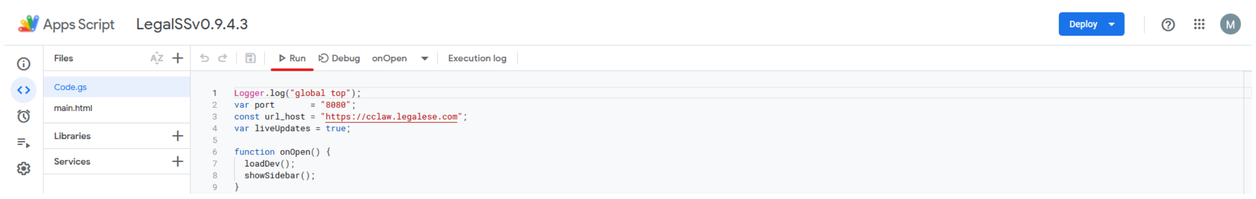


```
Logger.log("global top");
var port       = "8080";
const url_host = "https://cclaw.legalese.com";
var liveUpdates = true;

function onOpen() {
  loadDev();
  showSidebar();
}

function testIP() {
  let url = "https://l2.io/ip.js?var=userIP"
  let response1 = UrlFetchApp.fetch(url);
  Logger.log(response1.getContentText());
  // testWindow = this.window.location.hostname;
  // eval(UrlFetchApp.fetch('https://cdn.jsdelivr.net/npm/dohjs@latest/dist/doh.min.js').getContentText());
  // const resolver = new doh.DohResolver('https://1.1.1.1/dns-query');
  // resolver.query(testWindow, 'A')
  //   .then(response => {
  //     response.answers.forEach(ans => Logger.log(ans.data));
  //   })
  //   .catch(err => Logger.log(err));
  // const execSync = require('child_process').execSync;
  // const output = execSync('ls', { encoding: 'utf-8' });  // the default is 'buffer'
  // Logger.log('Output was:\n', output);
  return null;
}

function loadDev() {
  let spreadsheet = SpreadsheetApp.getActiveSpreadsheet();
  let sheet = SpreadsheetApp.getActiveSheet();
  let range = sheet.getRange("A1:Z10").getDisplayValues();

  port = devPort(range) || "8080";
  liveUpdates = devScan(range, /live updates (true|false)/i) || true; if (liveUpdates.toString().toLowerCase() == "false") { liveUpdates = false }
  Logger.log("setting port to " + port);
  Logger.log("setting liveUpdates to " + liveUpdates);
}

function showSidebar() {
  let cachedUuid = saveUuid();
  let [spreadsheetId, sheetId] = getSsid();
  let workDirUrl = (url_wd() + cachedUuid + "/" + spreadsheetId + "/" + sheetId + "/");
```

3. When the Sheets IDE loads, go back to the LegalSS Google Sheet in your browser and select the tab "Quickstart" if you are not on the page already. Select the "Case Study: PDPA DBNO" sheet at the bottom of this window.

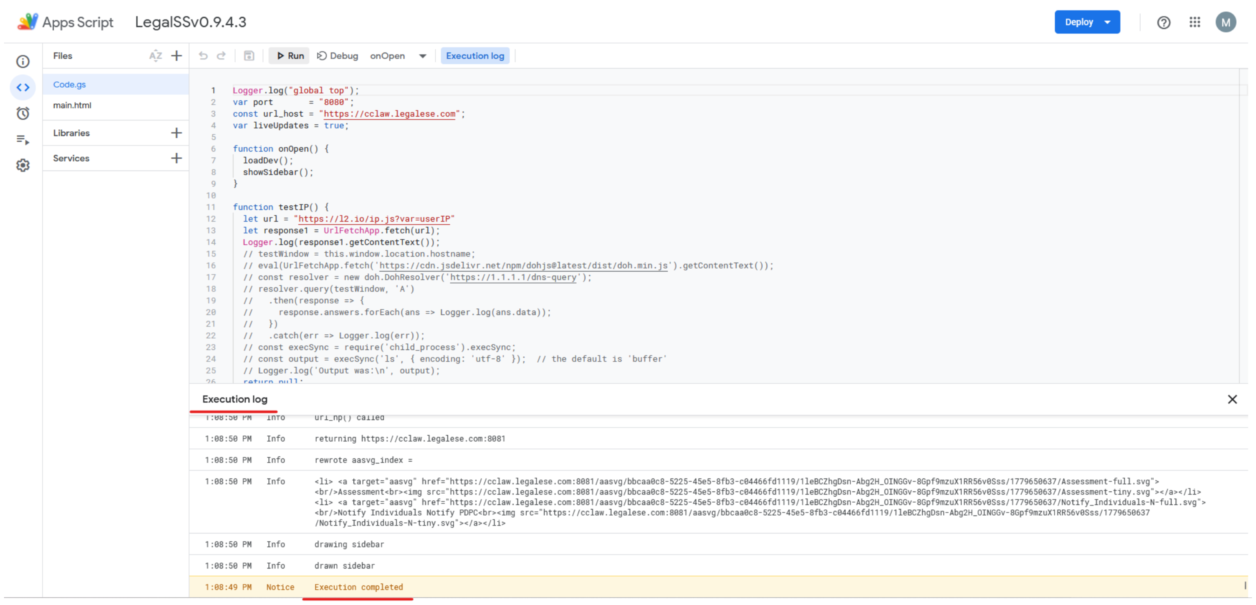


4. Back in the Sheets IDE, click "Run" to execute the "onOpen" function.



```
Logger.log("global top");
var port       = "8080";
const url_host = "https://cclaw.legalese.com";
var liveUpdates = true;

function onOpen() {
  loadDev();
  showSidebar();
}
```

An Execution Log should appear below the Sheets IDE. Wait until you read “Execution Completed”, highlighted in yellow, before you move on to the next step.

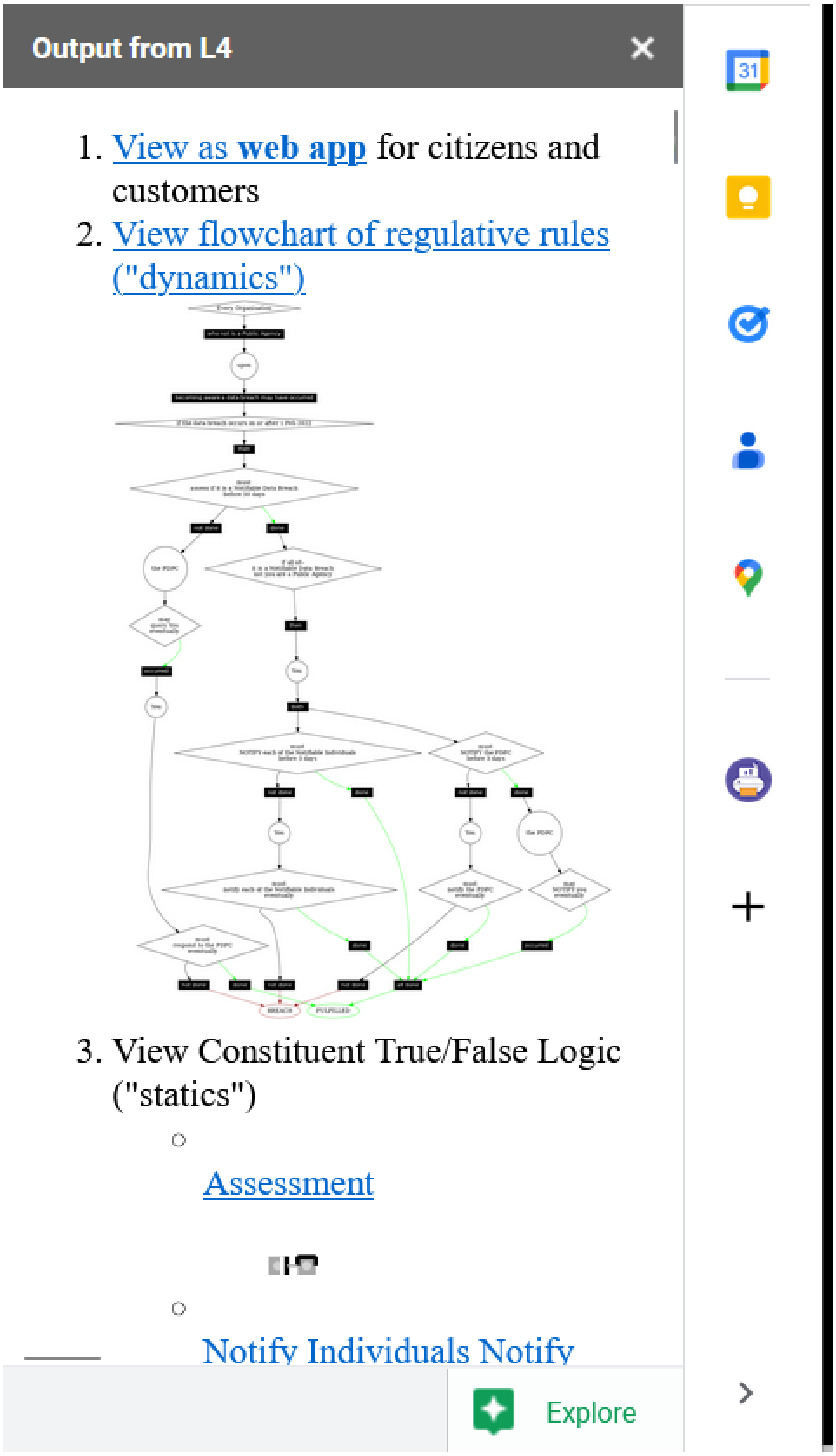


```
Logger.log("global top");
var port       = "8080";
const url_host = "https://cclaw.legalese.com";
var liveUpdates = true;

function onOpen() {
  loadDev();
  showSidebar();
}

function testIP() {
  let url = "https://l2.io/ip.js?var=userIP"
  let response1 = UrlFetchApp.fetch(url);
  Logger.log(response1.getContentText());
  // testWindow = this.window.location.hostname;
  // eval(UrlFetchApp.fetch('https://cdn.jsdelivr.net/npm/dohjs@latest/dist/doh.min.js').getContentText());
  // const resolver = new doh.DohResolver('https://1.1.1.1/dns-query');
  // resolver.query(testWindow, 'A')
  //   .then(response => {
  //     response.answers.forEach(ans => Logger.log(ans.data));
  //   })
  //   .catch(err => Logger.log(err));
  // const execSync = require('child_process').execSync;
  // const output = execSync('ls', { encoding: 'utf-8' });  // the default is 'buffer'
  // Logger.log('Output was:\n', output);
  return null;
```

Execution log

```
1:08:50 PM   Info     url_np() called
1:08:50 PM   Info     returning https://cclaw.legalese.com:8081
1:08:50 PM   Info     rewrote aasvg_index =
1:08:50 PM   Info     <li> <a target="aasvg" href="https://cclaw.legalese.com:8081/aasvg/bbcaa0c8-5225-45e5-8fb3-c04466fd1119/1leBCZhgDsn-Abg2H_OINGGv-8Gpf9mzuX1RR56v0Sss/1779650637/Assessment-full.svg">
                      <br/>Assessment<br><img src="https://cclaw.legalese.com:8081/aasvg/bbcaa0c8-5225-45e5-8fb3-c04466fd1119/1leBCZhgDsn-Abg2H_OINGGv-8Gpf9mzuX1RR56v0Sss/1779650637/Assessment-tiny.svg"></a></li>
                      <li> <a target="aasvg" href="https://cclaw.legalese.com:8081/aasvg/bbcaa0c8-5225-45e5-8fb3-c04466fd1119/1leBCZhgDsn-Abg2H_OINGGv-8Gpf9mzuX1RR56v0Sss/1779650637/Notify_Individuals-N-full.svg">
                      <br/>Notify Individuals Notify PDPC<br><img src="https://cclaw.legalese.com:8081/aasvg/bbcaa0c8-5225-45e5-8fb3-c04466fd1119/1leBCZhgDsn-Abg2H_OINGGv-8Gpf9mzuX1RR56v0Sss/1779650637
                      /Notify_Individuals-N-tiny.svg"></a></li>
1:08:50 PM   Info     drawing sidebar
1:08:50 PM   Info     drawn sidebar
1:08:49 PM   Notice   Execution completed
```

5. Return to the LegalSS Spreadsheet. You should see a sidebar appear on the right side of the page.


Every Organisation
who not is a Public Agency
upon
becoming aware a data breach may have occurred
if the data breach occurs on or after 1 Feb 2022
then
must assess if it is a Notifiable Data Breach before 30 days
not done
done
the PDPC
You
both
must NOTIFY each of the Notifiable Individuals before 3 days
must NOTIFY the PDPC before 3 days
occurred
BREACH
FULFILLED
Explore

Output from L4

1. View as **web app** for citizens and customers
2. View flowchart of regulative rules ("dynamics")
3. View Constituent True/False Logic ("statics")
   - Assessment
   - Notify Individuals Notify

If you do not see a sidebar, contact the L4 developers for help.

CHAPTER

THREE

DECLARATIONS AND DEFINITIONS

This chapter introduces a handful of keywords.

3.1 DECLARE and DEFINE, for data types and values, and HAS-Attributes

DECLARE and DEFINE have to do with data types and values.

If you are familiar with Object-Oriented Programming (from languages like Python, Java, C++, or Javascript) you will find the DECLARE and DEFINE concepts familiar.

We use DECLARE to set up our classes, our records, our types, our schemas, our ontology, our templates.

We use DEFINE to instantiate those templates with concrete values: the specific variables of a particular agreement.

These declarations and definitions are automatically exported to the programming language of your choice, lessening the burden of programming downstream. Some call this “”model-driven engineering””; others, “”low-code”.

Consider the following code

```
Type Declaration    ::=          DECLARE             MultiTerm           ␣
↪     [Type Signature]
                                 [Has-Attribute  ]
                                 [       ...     ]

Has-Attribute       ::=          HAS                 MultiTerm           ␣
↪     [Type Signature]
                                 [       ...     ]
                                 [Has-Attribute  ]
```

This syntax rule means you can have multiple HAS-Attributes, listed on subsequent lines. For convenience, only the first HAS keyword is necessary; subsequent lines don’t need it.

HAS-Attributes can nest, such that one record declaration can contain another. For example:

```
Variable Definition ::=     DEFINE          Value Term          [Type Signature ]  ␣
↪   //class-object instantiation
                            HAS             MultiTerm           [Type Signature ]
                                            [ ... ]
```

Variable definitions with the DEFINE keyword follow the same format as DECLARE.

We’ll talk more about the elementary data-types of L4 later: sum types, product types, lists, and dictionaries. We’ll also talk about type inference and type checking.

3.2 BY and WITHIN for Temporal Constraints such as Deadlines

The BY and WITHIN keywords set deadlines

```
Temporal Constraint ::= (BEFORE | AFTER | BY | WITHIN | UNTIL) Temporal Spec
```

A regulative rule without a temporal constraint is incomplete. L4 substitutes “EVENTUALLY” but will issue a warning so you are conscious that a deadline is missing.

3.3 MUST, SHANT, and MAY for obligations and permissions

Laws and contracts impose *obligations* and *prohibitions* on persons, and grant *permissions*. These ideas are central to *deontic logic*, and underlie L4’s keywords MUST, SHANT, and MAY, respectively.

```
Deontic Keyword ::= (MUST | MAY | SHANT)
```

Within the context of a single rule, these deontic keywords specify different consequences for the satisfaction or violation of the rule.

3.4 FULFILLED and BREACH for consequences in L4

The two fundamental consequences in L4 are FULFILLED and BREACH.

```
            If the actor does not perform the action by the deadline            If the␣
↪actor performs the action by the deadline
MUST                BREACHED                                                           ␣
↪                    FULFILLED
SHANT               FULFILLED                                                          ␣
↪                    BREACHED
MAY                 FULFILLED                                                          ␣
↪                    FULFILLED
```

We observe that a MAY rule is permissive: if you do it, fine! If you don’t, fine!

L4’s workflow diagrams follow a convention: a rule that is satisfied proceeds to the bottom right, while a rule that is violated proceeds to the bottom left. The “”happy path”” therefore runs along the right side of a diagram.

A MAY rule shows action to the right, and inaction to the left.

3.5 HENCE and LEST for regulative rules and connecting blocks of code

Ordinary programming languages use the IF … THEN … ELSE construct to connect blocks of code, based on whether the conditions in the IF were met.

L4 uses HENCE instead of THEN, and LEST instead of ELSE, to connect regulative rules, based on whether the preceding rule was satisfied.

```
Regulative Connector ::=    (HENCE | LEST)
                        Rule Label | Regulative Rule
```

Individual regulative rules connect with one another to form a graph, or a flowchart, describing a workflow.

3.6 The Semantics of rules

What are the semantics of a rule?

```
[Attribute Constraint   ]
[Conditional Constraint ]
[Upon Trigger           ]
[HENCE                             Rule Label | Regulative Rule ]
[LEST                              Rule Label | Regulative Rule ]
[WHERE                             Constitutive Rule
                                   [   ...     ]                ]
```

CHAPTER

FOUR

EXAMPLES IN L4 AND THE CONCEPTS AND KEYWORDS INTRODUCED IN EACH EXAMPLE

4.1 Must sing

Link to 'Must Sing' example

This example gives an overview of how to write a simple rule in L4 using the simple rule: "Every person who walks and eats or drinks must sing". The example illustrates key concepts along the way.

We thank Matthew Waddington for originally authoring the case from which this example is drawn.

4.2 Rodents and Vermin

Link to 'Rodents and Vermin' example

This example focus on a single decision rule drawn from a home insurance policy and its transformations to more easily understood forms.

Decisions express first-order logic, functions, predicates, judgements, and calculation in general.

Concepts introduced:

1. Boolean Structures in detail.
2. Visualization as an electrical circuit diagram.

Keywords introduced:

- DECIDE
- WHEN
- UNLESS
- AND
- OR
- NOT

4.3 Contract as Automaton

Link to 'Contract as Automaton' example

A loan is repaid in two instalments. The borrower has to stay out of trouble.

Concepts introduced:

1. Events and consequences
2. Obligations vs permissions
3. Process workflow diagrams

Keywords introduced:

- DECLARE
- DEFINE
- HAS
- IS A
- DO
- HENCE
- LEST
- MAY
- BY
- WITHIN

4.4 Case Study: Motor Insurance

Link to the case study 'Motor Insurance' example

Entity Relations, Ontology Inference, and Convenient Syntax for Predicate Logic.

Concepts introduced:

1. Combining regulative and constitutive rules
2. Guards in state transitions

Keywords introduced:

- DECIDE
- UNLESS
- WHO
- WHICH
- WHEN
- IF
- TYPICALLY

4.5 Case Study: PDPA DBNO

Link to the case study 'PDPA DBNO' example

L4 automatically generates a web app from real-world legislation & regulation. It is an encoding of a fragment of real-world legislation.

Concepts introduced:

1. Reference and Expansion
2. Temporal Keywords
3. State transitions

Keywords introduced:

- DECIDE
- UNLESS
- WHO
- WHICH
- WHEN
- IF
- TYPICALLY

4.5.1 Petri Net representation of PDPA DBNO

This section is still being worked on. We will continue our examination of the PDPA DBNO case with a deep dive into Petri Nets; it is intended to be a Petri Net representation of the PDPA DBNO example.

Concepts introduced:

1. Workflow diagrams in detail
2. BPMN used in industry
3. Process algebras

Keywords introduced:

- HENCE

CHAPTER FIVE

WORK IN PROGRESS

5.1 Logic Circuits

We will continue our examination of the PDPA DBNO case with a Boolean-Circuit representation of the examples found in *the examples page*.

5.2 Revisiting Logic Circuits

We will then return to the Rodents and Vermin example and introduce DMNs.

5.3 Revisiting Basic Syntax and Concepts

Afterwards, we will review the basic syntax and concepts found in L4.

5.4 Detailed Syntax Reference

A detailed syntax reference can be found in the L4 manual.

5.5 Model Checking

We will create a guide on using a model checker to search for loopholes and errors in your L4 code.

5.6 Type Checking and Inference

This will lead into an exploration of how the L4 interpreter performs type checking and type inference.

We will also explore Ontology inference and Other type inference.

5.7 Unit Testing

Finally, we will set up unit testing that can help set up scenario rules using the GIVEN and EXPECT keywords.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search