Liza Data Collection Framework v7.4.9

User & Developer Manual

R-T Specialty, LLC.

This manual is for the Liza Data Collection Framework, version 7.4.9.

This manual contains inline notes for developers of Liza.[1] *For an index of notes, see* *[Developer Notes Index], page 41*.

[1] To disable for user documentation, pass `--disable-devnotes` to `configure`.

Table of Contents

1 Design & Architecture

Liza is fundamentally a data collection framework— a fancy form for collecting, validating, and lightly processing user data.

The main components of the system are:

Assertions Basic validations against bucket data, producing errors and manipulating control flow. Invokes triggers to manipulate the UI and document. Assertions are compiled from Program sources. See Chapter 2 [Assertions], page 3.

Bucket The key/value store into which all document data are stored. Supports staging and rollback of data, processing deltas, and provides hooks that drive the rest of the system. See Chapter 3 [Bucket], page 4.

Client Basic logic for navigating between steps, prompting for user actions, display help text and basic document data, communicate with server, etc. See Chapter 4 [Client], page 7.

Data API Declarative abstraction for accessing and processing remote data (e.g. a RESTful service). See Chapter 5 [Data API], page 9.

Developer Dialog
Renders information about the system for debugging the client. Can monitor the bucket, assertions, classification results, and provides other useful features.

Predicate System
Processes classification data from external classifiers to determine applicability of specific questions. These data are used to determine what assertions are performed, what questions and groups display, and more. See Chapter 6 [Predicate System], page 12.

Program Internal representation of the Program with delegation of events to the assertion system. Contains compiled representation of all steps, groups, questions, assertions, metadata, and others. See Chapter 7 [Program], page 13.

Program UI
Rendering of elements specific to Programs, such as steps, groups, and questions. This is the equivalent of an HTML form. Directly monitors the bucket to perform UI updates. See Section 7.1 [Program UI], page 13.

Program XML
The source code for a Program, in XML format. See Section 7.2 [Program XML], page 15.

Server Provides REST API for serving Programs; saving data; revalidating, filtering, and recalculating data; and other types of processing. Code is shared with the client, ensuring identical behavior for appropriate behaviors. See Chapter 8 [Server], page 19.

Type Validation
Validates and formats bucket values for specific field (question) types. For example, a date field must be in a recognized date format, and will be normalized for display. See Chapter 9 [Validation], page 23.

More information about each can be found in their respective chapter/section.

2 Assertions

This system has maintenance concerns.[1]

There isn't much here yet. Maybe you can help?

[1] Assertions are compiled from the Program XML (see Section 7.2 [Program XML], page 15). Rather than using a library, it compiles a mess of largely duplicate code inline. This system needs to be *replaced*, not modified.

A replacement can either be in the form of a library (removing most if not all code generation from the Program XML compiler), or possibly compile into classifications and use the classification system. *The latter option is preferred, and would be more powerful with less maintenance.*

3 Bucket

There isn't much here yet. Maybe you can help?

3.1 Bucket Value Assignment

There isn't much here yet. Maybe you can help?

3.2 Bucket Diff

Changes to the bucket are represented by an array with certain conventions:

1. A change to some index k is represented by the same index k in the diff.

2. A value of `undefined` indicates that the respective index has not changed. Holes in the array (indexes not assigned any value) are treated in the same manner as `undefined`.

3. A `null` in the last index of the vector marks a truncation point; it is used to delete one or more indexes. The vector will be truncated at that point. Any preceding `null` values are treated as if they were `undefined`.[1]

Diffs are only tracked at the vector (array of scalars) level— if there is a change to a nested structure assigned to an index, that index of the outer vector will be tracked as modified. It will, however, recursively compare nested changes to determine if a modification *has taken place*.[2] Examples appear in Figure 3.1.

[1] The reason for this seemingly redundant (and inconvenient) choice is that JSON encodes `undefined` values as `null`. Consequently, when serializing a diff, `undefined`s are lost. To address this, any `null` that is not in the tail position is treated as `undefined`. We cannot truncate at the first null, because '`[null,null,null]`' may actually represent '`[undefined,undefined,null]`'.

[2] See `StagingBucket` method `#_parseChanges`.

Original	Diff	Interpretation
`'["foo", "bar"]'`	`'["baz", "quux"]'`	Index 0 changed to '`baz`'. Index 1 changed to '`quux`'.
`'["foo", "bar"]'`	`'[undefined, "quux"]'`	Index 0 did not change. Index 1 changed to '`quux`'.
`'["foo", "bar"]'`	`'[, "quux"]'`	Index 0 did not change. Index 1 changed to '`quux`'.
`'["foo", "bar"]'`	`'["baz", null]'`	Index 0 changed to '`baz`'. Index 1 was removed.
`'["foo", "bar", "baz"]'`	`'[undefined, null]'`	Index 0 was not changed. Index 1 was removed. Index 2 was removed.
`'["foo", "bar", "baz"]'`	`'[null, undefined, null]'`	Index 0 was not changed. Index 1 was not changed. Index 2 was removed.
`'["foo", "bar", "baz"]'`	`'[null, null, null]'`	Index 0 was not changed. Index 1 was not changed. Index 2 was removed.

Figure 3.1: Bucket diff examples.

Diffs are generated by `StagingBucket`. `null` truncation is understood both by `StagingBucket` and by `QuoteDataBucket`. A diff is applied to the underlying bucket by invoking `StagingBucket#commit`.

3.3 Calculated Values

There isn't much here yet. Maybe you can help?

3.4 Metabucket

The *metabucket* is a bucket-like key/value store separate from the data bucket.[3] It should be used to save data that should be accessible only to the server, but never the client.

Data must still be formatted as a vector, but unlike the data Bucket, vector values are sometimes structured data instead of strings.

A standard still needs to be devised to provide guidance for when storing structured data is appropriate, rather than a vector of strings.

The client has no means by which to access the metabucket. Custom fields can be populated by server-side DataAPIs (see Section 8.4 [Server-Side Data API Calls], page 21).

[3] It is stored in the `meta` field on the Mongo document.

Any fields prefixed with the string '`liza_`' are reserved and are populated automatically by the Server. They are shown in Table 3.1.

`liza_timestamp_initial_rated`
: A Unix timestamp representing the first time a document was acted upon by a rating service. This value is set once and is never updated or cleared.

Table 3.1: Metabucket fields populated automatically by the Server

4 Client

This system has maintenance concerns.[1]

There isn't much here yet. Maybe you can help?

4.1 Error Handling

This system has maintenance concerns.[2]

There are three layers of error checking:[3]

1. Required field checking— whether all required questions have been answered.
2. Type Validation— verify that questions contain valid data according to their declared type. Chapter 9 [Validation], page 23.
3. Assertions— arbitrary checks on data. Chapter 2 [Assertions], page 3.

Required fields fail serially— the system will notify the user of the required field, and direct him/her to it (usually through scrolling). A field is marked as *fixed* according to the rules in Section 4.1.1 [Managing Error State], page 7.

4.1.1 Managing Error State

Each failure caused by assertions is associated with a *failure stack*. The stack represents the trail of assertions that have run, containing the ids of all values asserted against. When any field or classification changes that is represented on the failure stack, the failure for the failed field associated with that failure stack is cleared.

Example: If an assertion for some question *foo* first checked the value of bucket field *bar*, and within its failure checked the value *c:predicate*, the failure stack would contain both of those ids. If either *bar* or the *predicate* classification changed, the question *foo* would have its error cleared.

Error state is managed by `ValidStateMonitor`.

4.2 Saving to Server

There isn't much here yet. Maybe you can help?

To save changes, the client posts only the bucket diff (see Section 3.2 [Bucket Diff], page 4) to the Server (see Chapter 8 [Server], page 19). Because JSON serialization encodes `undefined` values as `null` (as noted in Section 3.2 [Bucket Diff], page 4), and only the null in the tail position marks the truncation point, the Client first truncates the array to include only the first `null`.[4] An example is shown in Figure 4.1.

---

[1] The client is largely managed by a single class, `Client`, which has grown out of control. `Client` mediates essentially the entire system. Code is to be extracted out of this class as it is touched.

The other system mammoth is `Ui` (see Section 7.1 [Program UI], page 13).

[2] The complexity of this system and integration into legacy layers has caused maintenance trouble in the past. Each of the error checking layers need to be integrated to reduce complexity.

[3] Primarily for legacy reasons. They are being consolodated as the system is touched.

[4] The server would otherwise remove only the last index, even if multiple indexes were removed.

```
// given (two unchanged, three removed)
[ undefined, undefined, null, null, null ]

// encodes into JSON as (bad; represents four unchanged, one removed)
[ null, null, null, null, null ]

// Client truncates to (two unchanged, >=2 removed)
[ null, null, null ]
```

Figure 4.1: Client diff truncation

This conversion is handled by `XhttpQuoteTransport`. Examples can be found in the respective test case `XhttpQuoteTransport`.

5 Data API

This system has maintenance concerns.[1]

There isn't much here yet. Maybe you can help?

The *Data API* is a declarative abstraction for accessing and processing remote data (e.g. a RESTful service). The name stems from how it is used— to declare an remote API's inputs and outputs.

This system is generally used indirectly through the Section 7.2 [Program XML], page 15.[2]

All interaction with this system should be had through the `DataApiManager`.

The `DataApiManager` manages the entire operation— from triggering the initial request, to performing mapping, to populating bucket data. It takes only a `DataApiFactory` and Data API definitions.

Definitions have the following schema:[3]

```
{
  "type": "string",
  "source": "string",
  "method": "string",
  "params": {
    ["string(name)"]: {
      "name": "string(name)",
      "default": {
        "type": "string",
        "value": "string"
      },
      ...
    },
  },
  "retvals": [ "string", ... ],
  "static": [
    {
      ["string(param)"]: "string",
      ...
    },
    ...
  ],
  "static_nonempty": boolean,
  "static_multiple": boolean
}
```

Each of the above fields are defined by:

---

[1] This is a complex system with too much logic lying in `DataApiManager` (having been extracted from its old home in `Program`).

[2] See 'Data API' in the Liza Program UI Compiler manual.

[3] There are poor design decisions that will likely persist indefinitely because of integration with other systems, so future extensions may be messy (especially in the case of '`retvals`').

`type` Any type supported by `DataApiFactory` (e.g. '`rest`').

`source` Type-specific source of data. For e.g. '`rest`', this is a URI.

`method` Type-specific method for interacting with the API. For e.g. '`rest`', this is an HTTP method.

`params` Key-value mapping of input parameter names (as received by '`source`') to their default values. These inputs must be populated by the caller at the time of the request.

`retvals` Array of fields returned by the data source.

`static` Static values to prepend to the returned data. This is often used for adding "please select" text, for example.

`static_nonempty`
Whether statics should be added when there is return data; Otherwise, they will be added only if the response yields no results.

`static_multiple`
Whether statics should be added only if multiple data are returned. For example, a "please select" is only useful if there is more than one option for the user to select from. When '`true`', this has the convenient side-effect of auto-selecting the only result.

An example definition appears in Figure 5.1

```
{
  "type": "rest",
  "source": "/foo/city",
  "method": "post",
  "params": {
    "getVal": {
      "name": "getVal",
      "default": {
        "type": "string",
        "value": "getCityOptions"
      }
    },
    "zipcode": {
      "name": "zipcode",
      "default": {
          "type": "ref",
          "value": ""
      }
    }
  },
  "retvals": [ "city", "id", "state", "county", "country" ],
  "static": [ {
    "city": "(Please Select)",
    "id": "",
    "state": "",
    "county": "",
    "country": ""
  } ],
  "static_nonempty": false,
  "static_multiple": true
},
```

Figure 5.1: Example Data API definition

6 Predicate System

This system has maintenance concerns.[1]

For a practical application of these concepts, see its use in the Program XML (see Section 7.2.2 [Specifying Predicates], page 17).

The *predicate system* determines the *applicability* of certain objects (like questions and assertions) by associating them with predicates. The domain of discourse (variables which may be quantified) is listed in Table 6.1.

What it means for some object to be applicable depends on the context.

Type	Prefix	Description
Classifications	*None*	Results of applying an external *classifier* to the bucket (see Chapter 3 [Bucket], page 4).
Bucket Truth Predicate	`q:`	Whether the given name in the bucket (see Chapter 3 [Bucket], page 4) is non-empty and non-zero. The prefix '`q:`' refers to its most common use case—questions (see Section 7.1 [Program UI], page 13).

Table 6.1: Predicate system domain of discourse

This system is limited to existential quantification over the domain of discourse. For other quantifiers and higher-order logic, defer to one of the systems that contributes to the domain of discourse, like the classifier.[2]

Predicates are usually specified in the Program XML (see Section 7.2.2 [Specifying Predicates], page 17) and compiled into the program (see Chapter 7 [Program], page 13).

[1] New programs (using the old incarnation of TAME) use the classifier embedded into the rater by TAME. Old ones, however, still use the *Global Classifier*. This system isn't as well tested as TAME's— which needs to work properly for the sake of calculating premium— and has suffered from a number of bugs in the past.

The solution is to migrate all programs to TAME and remove that old code.

[2] This is usually TAME. The Program XML also supports inline classifications with TAME's syntax (see Section 7.2.2 [Specifying Predicates], page 17).

7 Program

This system has maintenance concerns.[1]

The *Program* is a declarative representation of an entire system. It is the highest level of abstraction from a data perspective. The user observes and interacts with a Program using the Section 7.1 [Program UI], page 13.

Programs contain a lot of metadata that is not in a convenience human-readable (or modifiable) format, some of which are redundant. Programs are ideally compiled from a Section 7.2 [Program XML], page 15 document.

7.1 Program UI

This system has maintenance concerns.[2]

The *Program UI* renders a Chapter 7 [Program], page 13 as a form.

At the highest level, steps are rendered in a tab-like manner, above the main form content. A step contains groups, which in turn contain elements such as questions. Groups are delimited in some manner defined by their style (see Section 7.1.1 [Group Styles], page 13).

Questions are rendered as form fields. Any time the respective Chapter 3 [Bucket], page 4 field is changed, the form field is updated to reflect those changes, after having first been formatted with the appropriate validator (see Section 9.1 [Formatting Values], page 23). When a question is changed by the user, the value is expected to be propagated to the Bucket (see Section 3.1 [Bucket Assignment], page 4).

Navigation between steps can be done via the *Navigation Bar* above the step content, or using "Go Back" and "Continue" buttons at the foot of the step content.

A *Sidebar* is rendered adjacent to the step content. It displays the name of the Program, as well as configurable metadata (usually through the '`sidebar`' node of the Section 7.2 [Program XML], page 15). It also displays question help text (also configured through the XML) and any error messages (see Section 4.1 [Error Handling], page 7).

7.1.1 Group Styles

Portions of this system need refactoring.[3]

Groups support a number of *group styles* that determine how they are delimited from other groups (see Section 7.2.1 [Defining Groups], page 15); how the elements they contain are rendered and laid out; and how multiple indexes are displayed, added, and removed. A list of available styles is detailed in Table 7.1.

---

[1] The `Program` class was one of the first prototypes created, and has evolved poorly with the rest of the system. It is the base class for all compiled programs, and it glues together too many other systems with a terrible API and little to no encapsulation.

With that said, it is one of the least touched classes (thus its state); developers rarely have the need to touch `Program`.

[2] The `Ui` class, in addition to `Client` (see Chapter 4 [Client], page 7), represent the two monoliths of the system. This mediates all UI-related tasks, and still has far too many concerns with far too many dependencies. Code is to be extracted out of this class as it is touched.

[3] Some group styles still use jQuery; they should be modified to use modern formatters and Liza DOM abstractions (see `src/ui/field` and `src/ui/styler`).

Name	Description	Multi-Index?	Add/Remove Index?
'`default`'	Groups are unstyled by default— they render elements as flat fields like a traditional form. Only the first index of elements is rendered.	N	N
'`accordion`'	Styled as a '`stacked`' group with a header that toggles the body associated with that index. When collapsed, only the header is visible for that index. The default styling indicates the collapsed status using an arrow in the header.	Y	N
'`collapsetable`'	Renders element label in the leftmost column like '`sidetable`'. Indexes are groups of rows delimited by headings, which collapse the respective group of rows when clicked.	Y	Add
'`sidetable`'	Renders elements as rows with label in the leftmost column rather than the top row. Each index is rendered as a column.	Y	Add
'`stacked`'	Groups respective indexes of elements such that one set of indexes appears atop of another set, much like separate groups are placed.	Y	N
'`tabbedblock`'	Each group is rendered as a block, with each index rendered as a tab to the right of it. Clicking a tab toggles the body content to the associated index. Elements are rendered within the box.	Y	N
'`tabbed`'	Like '`default`', but each index has a tab at the top of the group. Clicking a tab toggles the body content to the associated index.	Y	Y
'`table`'	A vanilla table with elements as columns, their labels across the top row. Each index is rendered in its own row.	Y	Y

Table 7.1: Group styles and index support

7.1.2 DOM Abstraction

jQuery is still used throughout parts of the framework and is a performance bottleneck— it needs to be fully removed and replaced with this DOM abstraction.[4]

[4] See `src/ui/ElementStyler`.

Liza was conceived long before frameworks like React existed. The implementation originally used Dojo because of its broad widget set, but it was later dropped because of extreme performance issues, especially on the browsers of the day (Liza had to support Internet Explorer 6!); at one point, certain steps took over a minute to load for the most unfortunate of users. jQuery was then used for various parts of the UI and for ease of DOM manipulation, because of the lack of good and universal DOM APIs back then. It too became a bottleneck. Using DOM APIs is now easy with modern browsers.

Liza's DOM abstraction contains a couple of components:

- *DOM Fields* represent a field on the DOM. Each field has a name and an index associated with the DOM node. Nodes are cached in memory after first access and queue requests during lookups to prevent stampeding. Provides basic DOM operations, including styling, containing row, and parent/sibling selecting. See `*DOM Context* is a slice of the DOM used for restricting queries. It can attach and detach sections of the DOM` and be further split into a context hierarchy. The `DomContext` provides field querying (see `DomField`) and caching. See `src/ui/context`.

It is important to always use these abstractions for any portions of the DOM under control of this abstraction; otherwise, assumptions used for caching may result in unintended behavior.

Using DOM contexts, DOM operations can be restricted to small windows (for example, groups or tabs), further reducing the impact of DOM queries.

The *root context* is represented by `RootDomContext`— sub-contexts can be obtained by invoking `#slice` on any context, which creates a new context from a subset of the parent. Detaching a parent will detach all child contexts.

Contexts can be manipulated in memory before being re-attached. Detach a context from the DOM with `#detach`, and attach with `#attach`. A context is aware of its parent and will re-attach itself to the DOM in the correct place. A child context always attaches to the parent, and so will not be rendered until the parent attaches.

Always detach from the DOM before performing extensive manipulations; this prevents the need for expensive re-painting until manipulation is done, at which point the context can be re-attached.

7.1.2.1 Field Styling

There isn't much here yet. Maybe you can help?

`DomField` is styled using field stylers (see `src/ui/styler`). The two most notable stylers are `and src/ui/field/NaFieldStyler` which style fields in error and hide fields that are no applicable respectively.

7.2 Program XML

There isn't much here yet. Maybe you can help?

7.2.1 Defining Groups

A *group* organizes and relates entities. If given a '`@title`', a header will be displayed above the group with that title. A group may optionally be given a unique identifier '`@id`', which is useful for debugging and scripting; any such identifier should be `snake_case`.

```
<group title="Foo Group">
  <question id="question_1" ... />
  <question id="question_2" ... />
</group>
```

Figure 7.1: Defining a simple group with a title

Groups can be independently styled in a number of different ways to provide different data representations (see Section 7.1.1 [Group Styles], page 13). Style is optional— the '`default`' group displays only the first index of each entity. Further styling can be done using CSS using '`@class`'.

All questions within a group share the same number of indexes; this is accomplished by monitoring the *group leader*. By default, the leader is the first indexable entity in the group (question, answer, or display); this can be overridden with the '`@indexedBy`' attribute. This attribute is only practically meaningful if the chosen group style supports indexes (see Section 7.1.1 [Group Styles], page 13).

```
<group id="leader_override" indexedBy="question_2" style="tabbed">
  <question id="question_1" ... />
  <question id="question_2" ... />
</group>
```

Figure 7.2: Overriding group leader using '`@indexedBy`'

Some group styles allow the user to add indexes; set '`@locked`' to '`true`' to suppress this feature.

7.2.1.1 Linking Groups

Data collection for similar entities may span multiple steps or groups; for example, one group may allow the user to define their risk locations, and a future group may ask for additional information for each of those locations. To have all entities within each of those groups share index length, they may be linked.

A *group link* is an arbitrary name given to a set of groups. Each group that wants to be part of the same link must set '`@link`' to the same string. The name of the link does not matter— it is *not* a reference to a group '`@id`'.

```
<group id="location" link="locations" style="tabbed">
  <question id="address" ... />
  <question id="city" ... />
</group>

<group id="underwriting" link="locations" style="tabbed">
  <question id="diving_board" ... />
  <question id="rabid_dog" ... />
</group>
```

Figure 7.3: Linking groups using '`@link`'

In Figure 7.3, each question in *location* and *underwriting* will have the same number of indexes— any time a new index is added to *location*, *underwriting* questions too will gain

another index and vice versa. There is no limit to the number of groups that can share the same link.

Linked groups are implemented such that the union of all fields in each of the groups of a given link are assigned to each of the individual groups. When the leader of any group changes, a new index is initialized for each group field, which (in the case of linked groups) is comprised of all fields in the link.

7.2.2 Specifying Predicates

Object predicates (see Chapter 6 [Predicate System], page 12) are specified using the '`@when`' attribute of certain nodes. It must contain a string of references understood by the system (see domain of discourse, Chapter 6 [Predicate System], page 12), all of which much match for the predicate to be true.

```
<question id="describe" type="noyes"
          label="Any special notes for this location?" />

<question id="vacant_desc" type="textarea"
          when="q:describe vacant property"
          label="Show only when a vacant property with the
                 question 'describe' non-empty and non-zero" />
```

Figure 7.4: Using the '`@when`' attribute

In Figure 7.4 above, question '`vacant_desc`' will be applicable when *all* of the values of '`vacant`', '`property`', and '`q:describe`' are true.[5] Within the context of the Program XML, this concretely means that the classifications '`vacant`' and '`property`' are true, and that the question '`describe`' is answered "yes". It reads as a sentence: "'`vacant_desc`'" is applicable when we should "describe a vacant property".

7.3 Document Metadata

Document metadata are metadata that describe certain aspects of the document; they are stored adjacent to the bucket in '`meta`' on the document root.[6] They should be used in place of a bucket field any time the client has no business knowing about the data. The '`meta`' record is called the *Metabucket*.

Metadata in the Metabucket should *not* be directly populated by external systems—Data APIintegration should be used instead (see below).

Metadata can be populated using any Data API(see Chapter 5 [Data API], page 9)—return data populate the Metabucket in the same way that they populate the Bucket. Definitions are stored in `meta.fields`, as shown in Figure 7.5.

---

[5] See Chapter 6 [Predicate System], page 12 for what "true" means for a particular variable in the domain of discourse.

[6] Terminology note: "document" and "quote" are the same thing; the latter is transitioning to the former for generality.

```
"fields":{
  ["string(name)": {
    "desc": "string",
    "dapi": {
      "name": "string",
      "map": {
        "string(dest field)": "string(source field)"
      }
    }
  }
}
```

Figure 7.5: Format of `meta.fields`.

Further, a key-value mapping of all bucket fields that— when modified, need to result in a metadata API call— are stored in the `mapis` object; this is shown in Figure 7.6.

```
"mapis":{
  ["string(field name)"]: [ "string(dapi name)", ... ]
}
```

Figure 7.6: Format of `mapis`.

8 Liza Server

This system has maintenance concerns.[1]

There isn't much here yet. Maybe you can help?

The *server*[2] is a RESTful service that serves as the HTTP server. It is designed to run under Node.js, motivated by the benefits of sharing code with the Client (see Chapter 4 [Client], page 7). The daemon is handled by the abstract `Daemon` monolith, which requires that a concrete `#getEncryptionService` method be defined by a subtype or trait. An example script to start the server is shown in Figure 8.1.

For local development, or to avoid use of any encryption service, use `DevDaemon`, which uses a dummy encryption service.

To start the server, invoke `bin/server`. You may also invoke `bin/server.js` directly using Node.js, but the use of `is recommended` as it uses the Node.js executable determined at configure-time, along with any command-line options required for Liza Server to function correctly. Additional options can be provided to Node.js using the *NODE_FLAGS* environment variable, which will be *appended* to the configure-time flags. This environment variable is *not* escaped or quoted, so be mindful of word expansion.

```
$ bin/server -c path/to/config.json

# providing additional options to node
$ NODE_FLAGS=--debug bin/server -c path/to/config.json
```

Figure 8.1: Starting the Liza Server

The HTTP server is managed by `http_server`.

8.1 Configuration

There isn't much here yet. Maybe you can help?

Liza is migrating to actual configuration file in place of environment variables. If no configuration is explicitly specified, it uses `conf/vanilla-server.json`.

Configuration loading is handled by `ConfLoader`. The configuration store `ConfStore` is asyncrhonous, so loading configuration from any external system is supported.[3]

8.2 HTTP Requests

There isn't much here yet. Maybe you can help?

Each HTTP request produces a `UserRequest` associated with a `UserSession`. Sessions are tightly coupled with PHP[4]; an existing PHP session is expected, as identified by the

[1] The `Daemon` monolith and `Server`, among other things, need refactoring. Quote initialization code should be moved into `ProgramInit`.

[2] Which may also be referenced as "quote server" in certain legacy contexts, referring to Liza's origin as an insurance rating system.

[3] Provided that you write the code to load from that system, that is.

[4] They don't have to be—refactoring is needed.

'PHPSESSID' cookie. Sessions are shared via Memcache (see ResilientMemcache).[5] If a session is not found (or is invalid), an HTTP 500 status code is returned and the HTTP request is aborted.

Requests are subject to a 120 second timeout, after which the request will be served an HTTP 408 status code. Note that this *does not stop background processing*— this timeout exists to prevent the user from hanging indefinitely.

If a process intends to perform background processing for any length of time (longer than a few seconds), it should complete the request as quickly as possible and use some other mechanism to report back progress (e.g. polling).

The UserRequest exposes raw request data with minor processing.

Path (#getUri)
: The path component of the URI. The method name is unfortunate.

Query data (#getGetData)
: Query string processed into a key/value object. Despite the name, this is also populated if non-GET requests contain query strings.

POST data (#getPostData)
: POST data processed into an object as if it were a query string (just as #getGetData). Since this requires data that is streamed asynchronously, this method takes a callback that waits for all data to become available; if the data are already available, it is immediately invoked with the processed POST data.

Cookies (#getCookies)
: Cookies parsed into a key/value object.

Remote address (#getRemoteAddr)
: IP address of the origin of the request. If the server is behind a proxy that sets the 'X-Forwarded-For' header, it is used instead.

Host address (#getHostAddr)
: Hostname of the server. If the server is behind a proxy that sets the 'X-Forwarded-Host' header, it is used instead.

Origin (#getOrigin)
: Origin of request. Only available if at lease one of the 'Origin' or 'Referer' headers are set. This is useful mainly for determining the protocol and host while behind a proxy.

User agent (#getUserAgent)
: The user agent string of the request.

Session ID (#getSessionId)
: The user's unique session id ('PHPSESSID').

Session ID name (#getSessionIdName)
: The name of the cookie from which the session ID originated (hard-coded to 'PHPSESSID').

TODO: Document return format and writing response data.

[5] Via a memcache session handler.

8.3 Posting Data

A diff of the bucket data (see Section 3.2 [Bucket Diff], page 4) is posted to the server on step save. This operation is performed asynchronously— the client need not wait for the step to save before the next can be requested.

Since validations are shared between the server and the client (see Chapter 9 [Validation], page 23), saving should only fail in exception situations. Should a failure occur, the server will instruct the client to kick the user back to the previous step (*kickback*).

A step cannot be saved if it is locked; such attempts will result in an error.

To prevent a user from skipping steps, the client may post only one step past the last step that has successfully saved; otherwise, the user is kicked back to the last step that was saved.

Once those basic checks have passed, the document is updated:

1. The diff is first *sanitized* to strip out unknown fields, internal fields posted by non-internal users, and to filter fields on permitted characters;
2. The sanitized diff is then applied to the existing bucket on the document;
3. Calculated values marked for storage (see Section 3.3 [Calculated Values], page 5) are re-calculated on the server (the values posted by the client have already been discarded by the first step in this list);
4. Server-side Data API calls (see Chapter 5 [Data API], page 9) are triggered using the diff as input data and an empty bucket for response storage (see Section 8.4 [Server-Side Data API Calls], page 21);
5. The last premium calculation date is cleared (indicating that premiums are no longer valid);[6]
6. Data marked as sensitive is encrypted and the ciphertext written to the bucket in place of the plaintext (see Section 8.5 [Encryption Service], page 22);
7. The current step is incremented and the *top visited step* is set to the larger of the incremented step or the existing top visited step id; and then
8. The new document state and bucket data are written to the database.

8.4 Server-Side Data API Calls

This system has maintenance concerns.[7]

Server-side Data API calls (see Chapter 5 [Data API], page 9) are triggered on step save (see Section 8.3 [Posting Data], page 21) and are handled much like they are on the client. Such calls are made automatically only for document metadata. Results of sever-side calls are *not* written to the bucket and are therefore useful for data that the client should not be permitted to modify; it also allows data to be kept secret from the client.[8]

Data API results on the client can be mapped back to multiple bucket values; the server, however, has serious concerns with how data are propagated for data integrity and security

[6] This concept is tightly coupled with insurance; it should be factored out at some point.

[7] This makes use of `DapiMetaSource` to encapsulate the horrible API of `DataApiManager`; the latter needs cleanup to remove the former.

[8] All bucket data is served to the client, with the exception of internal fields if the user is non-internal.

reasons. Further, document metadata can be structured, unlike the Bucket which has a rigid matrix format (see Chapter 3 [Bucket], page 4). Therefore, the entire response is mapped into the parent field; defined return values are used only for filtering.

When a DataAPI request is made, it supercedes any previous requests that may still be pending for that same index.

8.5 Encryption Service

There isn't much here yet. Maybe you can help?

9 Validation

There isn't much here yet. Maybe you can help?

9.1 Formatting Values

There isn't much here yet. Maybe you can help?

10 Hacking Liza

There isn't much here yet. Maybe you can help?

This chapter provides general information and guidance for [prospective] developers of Liza.

For writing classes; interfaces; and traits, developers should familiarize themselves with GNU ease.js. For writing unit tests, developers should be familiarize themselves with Mocha and Chai. For more information on the libraries used by Liza, see Section 10.2 [Libraries], page 26.

Most source files have a general structure that must be followed. For example, all such files must have a copyright header and must be named after the class they define or system under test. For more information, see Section 10.1 [Source Files], page 24.

Generally speaking, developers should be familiar with vanilla ECMAScript; DOM APIs; and have a basic understanding of Node.js for well-rounded Liza development. Writing this manual requires basic understanding of Texinfo. References for these topics and others are provided in see Section 10.3 [Developer Resources], page 27.

10.1 Source Files

There isn't much here yet. Maybe you can help?

This section describes conventions for organizing files, both in directory structure and in content.

10.1.1 Copyright Header

Every source file should begin with a copyright header including the appropriate years and license information. This ensures that this information is always available even if the file becomes separated from the source distribution (e.g. is distributed independently). Further, it is necessary to indicate that the source file is distributed under the GNU General Public License version 3 *or later*— that "or later" clause does not exist as part of the license itself, and so the mere presence of the license in `COPYING` is insufficient.

The copyright headers vary slightly between JavaScript and Texinfo source files, represented in Figure 10.1 and Figure 10.2 respectively.

```
/**
 * DESCRIPTION OF FILE
 *
 *  Copyright (C) 2017, 2018 R-T Specialty, LLC.
 *
 *  This file is part of the Liza Data Collection Framework
 *
 *  Liza is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
```

Figure 10.1: Example copyright header for JavaScript files

```
@c  This document is part of the Liza Data Collection Framework manual.
@c  Copyright (C) 2018 R-T Specialty, LLC.
@c
@c    Permission is granted to copy, distribute and/or modify this document
@c    under the terms of the GNU Free Documentation License, Version 1.3
@c    or any later version published by the Free Software Foundation;
@c    with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
@c    Texts.  A copy of the license is included in the section entitled ``GNU
@c    Free Documentation License''.
```

Figure 10.2: Example copyright header for JavaScript files

For more information, see "How to Apply These Terms to Your New Programs" under the GNU General Public License version 3.

10.1.2 ECMAScript Strict Mode

ECMAScript 5's Strict Mode throws errors in more situations that may lead to buggy code, allows for better optimization of ECMAScript code during runtime, and prohibits syntax that conflicts with future ECMAScript versions. It also enables certain features, like using `let` inside blocks.

It should always be enabled going forward as shown in Figure 10.3. The statement should immediately follow the copyright header (see Section 10.1.1 [Copyright Header], page 24), before any other code.

```
// Copyright header

'use strict';

// ...
```

Figure 10.3: Enabling strict mode

10.2 Libraries Used

Liza does not use many libraries. The primary reason for this was that few libraries useful to Liza existed during its initial development— Node.js and its community was still very young. With that said, care should be taken to ensure that libraries are added only after a careful analysis of its costs and benefits, as they add volatility to the whole system and may also introduce security vulnerabilities outside of our control. They further introduce maintenance obligations for keeping up with newer versions of those libraries and addressing backwards-compatibility concerns.

10.2.1 System Libraries

Liza was originally developed using JavaScript (first ECMAScript 3, and then ECMAScript 5). JavaScript does not natively support the classical object-oriented model familiar to users of more traditional classical object-oriented languages like Java, C++, C#, and PHP. Liza is built using GNU ease.js, which provides those familiar features. The primary language used by developers in the office that created Liza is PHP, which motivated the creation of ease.js to ease the burden of entry.

Consequently, Liza is written in a classical object-oriented style rather than using prototypes. The `class` keyword introduced in ECMAScript is largely syntatic sugar around the prototype model and does not address the primary concerns of ease.js, nor does it provide traits.

The project is now migrating toward TypeScript, so new code should not use ease.js unless required and an effort should be made to move existing code away from ease.js. For more information on this migration, see See Section 10.4 [TypeScript Migration], page 27.

10.2.2 Testing Libraries

Mocha is used as the test runner for JavaScript unit tests. Chai is the assertion library. This differs from PHP development where a single system (PHPUnit) encompasses both of these needs.

Chai offers a few different styles of assertions ("should", "expect", and "assert"); Liza uses "expect".

A library to aid in mocking TypeScript classes needs to be researched.

10.2.3 UI Libraries

jQuery was used in the past, but has been largely purged from the system (and continues to be removed) due to strong performance issues. Further, now that browser APIs have stabalized and Liza no longer needs to support as far back as Internet Explorer 6, the standard DOM APIs are more than sufficient.

Liza instead provides its own UI and DOM abstractions (`src/ui`) that have been optimized for Liza's needs.

There are modern frameworks that may overlap with the type of UI operations that Liza performs, as well as certain DOM optimizations that it performs; however, it is unlikely that such frameworks (e.g. React, Angular, Meteor) will ever be integrated, as the cost of doing so exceeds the marginal benefit.

10.3 Developer Resources

MDN[1] is an essential resource for web development in general, especially for JavaScript/ECMAScript and the various Web APIs. It contains resources for all levels, including for those unfamiliar with JavaScript. All developers should familiarize themselves with the resources available on MDN so that they understand what type of information is readily accessible for future reference.

An overview of TypeScript can be found in its Handbook. The language borrows concepts from a number of others, so many concepts may be familiar to you. TypeScript uses structural typing (duck typing). In Liza, we also choose to implement nominal typing using "branding" (`src/types/misc.d.ts`). A language specification is also available.

The Server (see Chapter 8 [Server], page 19) uses Node.js. Although it's largely abstracted away, there may be times where you need to touch on it, in which case the Node.js documentation will be helpful. However, it is important to note the version of Node.js that Liza is currently using, as it may be woefully out of date and require looking at older versions of the documentation.

This manual is written using Texinfo, which is the documentation format of the GNU operating system. The format is structured and well-suited for software documentation with output in a variety of formats. Looking at the source code of this manual will be helpful—it provides the general structure and numerous macros that are specific to Liza.

Data are persisted using MongoDB. Database operations in Liza are abstracted away, but it's helpful to understand how to query the database directly to understand how the system works and composes its data, and for the purposes of debugging.

For information on specific libraries used by Liza, see Section 10.2 [Libraries], page 26.

10.4 TypeScript Migration

There isn't much here yet. Maybe you can help?

This section contains notes regarding a migration to TypeScript. It is intended to serve as a guide— it is not prescriptive.

10.4.1 Migrating Away From GNU ease.js

Liza was originally written in GNU ease.js. TypeScript now provides many features that ease.js was written to address, though not all (most notably traits).

Since ease.js was designed with JavaScript interoperability in mind, and TypeScript generates prototypes from classes, TypeScript classes serve as drop-in replacements under most

[1] Formerly the "Mozilla Developer Network"; see "The Future of MDN: A Focus on Web Docs" for the history of the rename.

circumstances. However, subtypes must be migrated at the same time as their parents, otherwise type checking in TypeScript cannot properly be performed. If this is a concern, type assertions can potentially be used to coerce types during a transition period in conjunction with ease.js' '`Class.isA`'.

Often times you will need to reference a class or interface as a dependency before it has been migrated away from ease.js. To do this, create a corresponding `.d.ts` file in the same directory as the dependency. For example, if a class `Foo` is contained in `Foo.js`, create a sibling `Foo.d.ts` file. For more information, see Declaration Files in the TypeScript handbook.

ease.js implements stackable Scala-like traits. Traits are *not* provided by TypeScript. Traits will therefore have to be refactored into, for example, decorators or strategies.

10.4.2 Structural Typing

TypeScript implements structural typing, also called duck typing. This means that any two types sharing the same "shape" are compatible with one-another.

For classes, this can be mitigated by defining private members, which then ensures that compatible types are indeed subtypes.

Interfaces can be used in either the traditional OOP sense, or as a means to define the shape of some arbitrary object. Since interfaces do not define implementation details, the distinction isn't important— it does not matter if we receive an instance of an object implementing an interface, or some object arbitrary that just happens to adhere to it.

In other instances where we want to distinguish between two values with otherwise compatible APIs, Nominal Typing below.

10.4.3 Nominal Typing

It is sometimes desirable to distinguish between two otherwise compatible types. Consider, for example, a user id and a Unix timestamp. Both are of type `number`, but it's desirable to ensure that one is not used where another is expected.

TypeScript doesn't directly support nominal typing, where compatibility of data types are determined by name. Liza uses a convention called "branding", abstracted behind a `NominalType` generic (defined in `src/types/misc.d.ts`).

```
type UnixTimestamp = NominalType<number, 'UnixTimestamp'>;
type Milliseconds  = NominalType<number, 'Milliseconds'>;

function timeElapsed( start: UnixTimestamp, end: UnixTimestamp ): Milliseconds
{
    return end - start;
}

const start = <UnixTimestamp>1571325570000;
const end   = <UnixTimestamp>1571514320000;

// this is okay
const elapsed = timeElapsed( start, end );

// this is not, since elapsed is of type Milliseconds
timeElapsed( start, elapsed );
```

Figure 10.4: Example of nominal typing

Consider the example in Figure 10.4. Both `UnixTimestamp` and `Milliseconds` are a `number` type, but they have been defined in such a way that their names are part of the type definition. Not only does the compiler prevent bugs caused from mixing data, but nominal types also help to make the code self-documenting.

If you want to have self-documenting types *without* employing nominal typing, use type aliases.

There are no prescriptive rules for whether a type should be defined nominally.

In some cases, it's useful to use nominal types after having validated data, so that the compiler can enforce that assumption from that point forward. This can be done using type assertions.

```
type PositiveInteger = NominalType<number, 'PositiveInteger'>;

const isPositiveInteger = ( n: number ): n is PositiveInteger => n > 0;

const lookupIndex<T>( arr: T[], i: PositiveInteger ): T => arr[ i ];

// untrusted input from the user
const user_input = readSomeValue();

if ( isPositiveInteger( user_input ) )
{
    // user_input is now of type PositiveInteger
    return lookupIndex( data, user_input );
}
```

Figure 10.5: Validating nominal types

In Figure 10.5 above, we only assume something to be a `PositiveInteger` after having checked its value. After that point, we can use TypeScript's type system to ensure at compile time that we are only using positive integers in certain contexts.

Never cast values (e.g. using '`<PositiveInteger>user_input`*') when type predicates are provided, since that undermines type safety.*

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Concept Index

S

T

U

V

Developer Notes Index