# Archipelago Specs and Features

**Updated:** August 2022
**Current Release Cycle:** 1.0.0
**Link to this Document:** https://tinyurl.com/ArchipelagoOneSpecs
**Latest Documentation:** https://github.com/esmero/archipelago-documentation/tree/1.0.0-RC3
**Previous RC Roadmap:** https://github.com/esmero/archipelago-deployment/issues/172
**Latest Roadmap:** https://github.com/esmero/archipelago-deployment/issues/190
**Official Deployment Strategies:** Dockerized environment (docker-compose)
https://github.com/esmero/archipelago-deployment/tree/1.0.0
https://github.com/esmero/archipelago-deployment-live/tree/1.0.0
**Additional Deployment Strategies:**
Bare Metal: https://devdbopen.byterfly.eu/doku.php?id=architecture

**Docker Containers:**

1. MySQL 8.0.28/MariaDB 10.6.8
2. Solr 8.11
3. Cantaloupe 6.0
4. PHP-FPM 8.0
5. NLP64 (Natural language Processing Server)
6. NGINX with Certbot/SSL self renewal
7. Min.io (Latest with Gateway Compat)
8. Redis 6.2

**Drupal Versions:** 9.4
**Setup/Config:** Sync folder with initial Drupal environment setup/config/yaml. Includes all configs to get started. AMI Set with Demo data updated for 1.0.0

## Archipelago Software/Modules

### strawberryfield v1.0.0

This is the core module of Archipelago and deals with Digital Objects definitions and Metadata capabilities. Concern is storage, logic and binding services.

Provides the following core Archipelago features:

- Strawberry field Type(SBF):

  Smart Drupal Field that can store complete JSON hierarchies (Descriptive and Technical Metadata) natively in a JSON Database BLOB and exposes via our Key Name Provider Plugin (UI) direct access to any internal properties/value to Drupal/Search/Solr (native to Drupal) with querying capabilities.This basically

means that any data/metadata can be merged/extracted/transformed and exposed to Search/Index/interaction directly via the UI. Even upfront before it even exists and changed anytime without schema updates.
    - SBF can manage direct JMESPATH queries against the stored JSON and cache the results.
    - SBF expose all data also flattened and a Dictionary of it's internal Keys
    - Any Drupal Content type needs only a single of these field types to be attached (Bundle) to be recognized and to be handled as Archipelago Digital Objects (**ADO**s).
    - SBF stores also Activity Stream Metadata to keep track of who created.update and how it was done. (e.g via a Webform, which one, or via AMI)
    - SBF generates automatic hierarchical Drupal taxonomies with every new JSON KEY added (Vocabulary Builder)
- Archipelago Digital Objects:
    - Any Node with a SBF attached is an ADO.
    - ADOs are simplistic Nodes and have special /do/{uuid} URLs instead of the /node/{numeric ID} pattern
    - ADOs trigger Events on Create/Edit/Update/Versioning/Delete
    - ADOs Store all File (attachments) references and metadata inside the SBF JSON
    - ADOs can handle **any** number of Different Files and Formats
    - ADOs expose Native Drupal Computed fields for “related Objects/Entity references e.g Collections” and for File entities too
    - ADOs via SBF can manage direct JMESPATH queries against it's JSON
    - ADOs self deposit its data/metadata as JSON files inside the permanent Storage. Means there are always 2 pure text representations on creation of every NODE. A full metadata JSON and a full Drupal Node dump (also JSON)
- File Entities:

    **File** referenced by an **ADO**’s JSON gets classified, identified and technical metadata extracted and merged with the main SBF JSON.
    - CHECKSUM/EXIF/PRONOM/IDENTIFY is run on every File
    - Technical metadata lives structured and classified by type (image, model, document) side by side with Descriptive Metadata
    - Files are sanitized and stored in Persistent Storage (Minio) using a simple checksum/hash structure/folder and prefixed for faster Cloud access
    - File storage Structure is plugable/hookable and can be extended to any desired hierarchy/pattern (OCFL, etc but does not ship in 1.0.0-RC2 with others)
    - File access is mediated by Archipelago, Drupal usage count too.
- Events and Subscribers:

    A full set of **ADO** and JSON based **events** are defined and **Event Subscriber** provided that deal with post processing **ADO**s in order. From **File** clean up to Identification, To labels based on Metadata, to deposit in storage, all happens via Event Subscribers. Any external Drupal module can tap into these and add extra functionality.

Performance Benchmarks can be enabled and is tracked on Subscribers too (e.g time processing on “save”)

- Strawberry Flavors:

  Special Data Source Definition to Index/Store/expose/Find Post Processed parts of files/ADOs. E.g for a single PDF, multiple HOCRs are generated. Each page generates a Strawberry Flavor that ends on its own, Drupal native Solr document pointing back to the **File** and **ADO** that generated it, it's sequence, who generated it (more on this in Strawberry Runners module), checksum, etc.

- HOCR Services:

  Dependent on Strawberry Flavors, exposes direct solr Highlights API/endpoints for HOCR highlights using **solr-ocrhighlighting** solr plugin (written by the Bavarian State Library, Archipelago is the first repository to use this in production outside of its institutional home) shipped with the deployment strategy. This means direct fast search for HOCR with coordinates without any pre or post processing.

- Hydroponics Service:

  Configurable Background processor for Drupal Queues. Triggers via CRON, drush or manually and does any heavy processing, e.g HOCR, WARC transformations, Batch Ingest. Works with **ReactPHP**, a real time timer/loop/multi child PHP capable of running extremely fast and efficiently. Has UI/UX reset, monitoring and allows configuring which Queues (inclusive not Archipelago ones only) will run. Wakes up. Works. Checks work left. Dies. So it does not waste any CPU/memory resources. It is also Multi Site capable and integrated with Drush 10.

- Key name provider Plugins:

  Because JSON can be complex and deeply hierarchical and Solr is not, we provide dynamic properties (using Typed Data API of Drupal) to expose internal JSON/KEYS/VALUEs to the native Drupal / Search API ecosystem. These Plugins provide different “extraction” and “exposure” capabilities for SBFs. You can Setup new ones with knowledge of existing data or based on future needs. Some Key Name Providers can take a full Vocabulary (schema.org) and use the properties there in “Expectation” that you use the same ones in your metadata. Others use JMESPATH queries to take values and expose them to Solr as multiple flat values, or join multiple sources in one (e.g Subjects from WIKIDATA and LoC inside a single Solr field). Others can take string/numeric values from your SBF JSON and cast them into real entities to allow parent/child references to be defined dynamically and index in Solr data from a Parent/Parent collection. New Key Name Providers are being built in each release.

- Drush 10 Commands:

  JSON API based Drush wrapper for ingestion of ADOs using a single JSON file (only the metadata) and folder with files. Wraps the JSON API, uploads files first, recovers UUIDS, enriches the JSON and ingests the New ADOs. UUIDs can be provided to avoid double ingest of the same source.

- Field Widget:

  Provides a basic RAW JSON edit widget that can be used by admins to quickly correct/edit the SBF JSON directly without any complex UI. It also does JSON validation.

## Webform_strawberryfield v1.0.0

This module handles Individual ADO Ingest/edit workflows via Drupal *Webforms*. It also wraps all our Linked data functionality and endpoints.  Concern is INPUT.

- Field Widgets

  Provides 2 Field Widgets for SBF. These widgets allow any *Webform* (even multistep) to be used as Input Workflow for new/existing ADOs. These widgets Dynamically load Ajax driven *Webforms* that can read/write JSON Metadata back and forth from SBF (ADOs). Each Webform can be customized and has settings that allow them to be used across multiple Form Modes. This gives the user a high flexibility to build differentiated ways of editing and creating nodes and even separate via permission roles on what can/not be added/edited.

- Webform handler

  To allow Drupal *Webforms* to read/write JSON and also serve as standalone URLs for submitting ADOs, this module provides a special *Webform* handler that deals with reading data from an ADO, populating the Webform and managing all the internals needed to fill back the ADO during ingest/edit. This Webform handler can be attached to any Webform to allow it to act as ingest/edit workflow using the provided Widgets or also as a standalone URL/endpoint for self deposit workflows. It also provides **automatic saving** capabilities (and deferred validation), so users can start an ingest, leave a session unfinished and come back before a week to complete the ingest.  Jumping between pages/steps is also allowed. This handler can also map Elements directly to an ADO SBF when not used via a Widget/Form Mode.

- Metadata Webform Elements

  This module provides additional elements to the built in Webform ones (50+) that can be used in the “Build” *Webform* interface for Metadata/Linked data needs to create a better Metadata edit/ingest experience. It provides

  - Wikidata autocomplete
  - Complete Set of LoC (anything + RDF based selections) autocompletes
  - Nominatim (Open StreetMaps) with free text query
  - Getty AAT (Fuzzy and exact) autocomplete
  - VIAF autocomplete
  - Panorama Tour Builder with multiple Scenes and Hotspots
  - Agents with Roles and LoC Autocomplete
  - XML Import to JSON
  - CSV Import to JSON
  - Strawberry Field Transplanter (Copies values from any place of the JSON into other *Webform* Elements dynamically)
  - Metadata aware Dates (with Ranges, Free Form, EDTF, ISO8601)
  - EXIF preview of uploaded Files
- Webform Values override/prefill

  Allows any Webform to be “pre populated” with data from an existing ADO via an URL get argument.

- LoD API endpoints

  All Vocabularies supported by default (and their variants) are exposed as URL/API endpoints. This can be used by other systems (or by archipelago itself) to reconcile “labels” against external authorities and give back the correct label/URI back, or a full list of candidates. All these endpoints are GET based and return cacheable JSON. They normalize the way of querying the different sources (some are JSON, REST, others Sparql) giving the user a single/simple way of asking for LoD.

## Format_strawberryfield v1.0.0

The role of this module is to extract and format the RAW JSON into any other representations, from interactive HTML, Viewers and JS plugins, to Schema based Metadata representations like IIIF Manifests, GeoJSON, DC, MODS, schema.org, etc (any). This module also provides most of the core IIIF integrations Archipelago has. Concern is OUTPUT.

- Field Formatters

  This Module provides a large set of Field Formatters (viewers and Metadata Casters) to cover most of the use cases we have encountered, new ones can be added by extending the base Plugins:

  - **IIIF Media Formatter:** based on Open Sea Dragon with Custom Javascript. It can take any referenced Images from a SBF and display those in an ADO Display or Views/Block based Display. It uses IIIF to request images, precomputes correct sizes and exposes them to Open Seadragon. It also binds with Annotorious and allows Direct Edit of WebAnnotations on any Viewer. Annotations can be edited, added, per image (when many too) and then Saved/persisted when going into edit mode
  - **Image Formatter:** simplest use case for images, takes an image file and outputs a configurable HTML image/link using IIIF.
  - **3D Formatter:** uses Three.js to Render 3D files like OBJ and STL files referenced in any SBF.
  - **Video Formatter:** uses HTML5 to render Video with subtitle capabilities
  - **Audio Formatter:** uses HTML5 to render audio with subtitle capabilities
  - **PDF Formatter:** simplistic PDF rendering using custom integrated pdf.js. Lacks the “fancy” functionality of the full version (on purpose) but can stream PDFs (even huge ones) in almost real time without any delay.
  - **Panorama Pannellum Formatter:** Can render 360 degree (and less degrees) Panorama Images and expose hotspots that can open ADOs and external URLS in a pop up. The same Formatter can also read Panorama Tours build (imagine a compound) of many other Panorama based ADOs and connections between scenes. It also has smart webGL capabilities limiting the IIIF image size to what each device can really handle a predictive (but simple) memory calculator to avoid calling a IIIF image size that does not fit in your Cantaloupe Server’s memory (e.g the 38Kx18K Panoramas of Nasa Perseverance Rover that require 2 Gbytes of RAM per call)
  - **Metadata Display (Twig) Formatter:** this is probably the most powerful of all Formatters. It takes a Metadata Display Entity (provided by this module) that holds a Twig template inside, pushes JSON and extra Context data into it and renders it in realtime. This allows users to build metadata listings, complex interfaces, blocks, etc, without having to modify the theme, or even embed JSON-LD or any other metadata directly into the Rendered Page. It's fast caching and very performant.
  - **Mirador Viewer Formatter (version 3):** This Formatter is very flexible. It can take a Metadata Display Entity (that produces a IIIF manifest v2 or v3), an Metadata Exposed endpoint (that produces a IIIF manifest v2 or v3), a json KEY with URLs that point to external IIIF manifests or even a list of other Nodes (also coming from the JSON metadata) to render complex multi canvas IIIF scenes. All these settings are configurable and

allow users, e.g to showcase comparisons of ADO files with external IIIF manifests, or a whole Children list of related ADOs. It has also Annotation Capabilities

- **Paged Formatter:** Implements a custom Internet Archive Bookreader JS viewer capable of reading from IIIF Manifests via an exposed Metadata Endpoint or a Metadata Display Entity that outputs a IIIF manifest (V2). A popular choice (Provided by default in each deployment) is to use this to render from PDF directly Pages as Images via IIIF. It also integrates perfectly with our HOCR endpoints and allows highlights. If no HOCR is processed yet for a Book, PDF or set of Images, it will hide the Search Field.
- **Replayweb Formatter:** This Formatters integrates (a collaboration with replay.web) an embedded Web Archive reader/player for WARC and WACZ files that uses JS Web Workers to render navigable and searchable Websites inside your ADO. It includes also a fully customizable selection strategy of which Files to use (e.g in case there are a WARC and a WACZ of the same content, being WACZ real time streaming) via JMESPATH. This extra config option is being ported to the 3D viewer/Panorama Viewer/Audio Viewer/etc because it allows users to define exactly which files they want to be exposed to any viewer, at metadata level. E.g: only select WebArchives files that are larger than 500 Mbytes, Updated in the last 2 weeks, that are of "application\/vnd.datapackage+zip" mime type.
- **Map Formatter:** Uses leaflet and custom settings to render maps. Users can select which tile sources they want to use, zoom level, max zoom levels, etc. Geographic Data is provided via a Metadata Display Entity that generates GeoJSON (Huge flexibility) using, e.g, the Nominatim Webform Element and can also read GEOJSON from external URLS referenced inside the SBF JSON.
- **Citation Formatter:** Uses twig modified input to generate via CiteProc any existing Citation format. Includes JS that allows you to change/view different ones at the same time.
- **Common functionality:** Every Formatter viewer can either use the Global (per site) IIIF Server or a custom one. Which allows complex scenarios where certain viewers are serving Images at lower resolutions with Watermarks while the same viewer for an Curator can expose the full size. Which JSON key provides the data is also configurable, the number of media to expose and even the width/height of each Viewer/Formatter. Also a JMESPATH selector allows an even finer grained option when multiple source match the basic scenario. Embargoes can be applied to any formatter (DATE and IP address), an alternative file display can be used when embargo is present and

Multiple files can be grouped together to generate a semantic unity. E.g a single Video + 2 subtitles uploaded to the same key will be assumed as being a group.

- Metadata Display Entity

Probably the most important part of Archipelago after the SBF. These special entities hold User facing/editable Twig templates. These Twig templates can be used to generate HTML/Display facing elements in each ADO or in Views but also Formal Metadata (Schema based). Each Metadata Display Entity also exposes it's desired output format (XML, JSON, HTML, TEXT, CSV, TURTLE, JSON-LD) and can also read fixed JSON (e.g a list of all available ISO Languages, things you would not want to ingest to every ADO) from a SBF added to the same Entity Type. These entities have “self rendering” capabilities and are Cache Tags aware, meaning if the Source data (a SBF JSON) does not change they are automatically returned from a previous cache. This makes huge IIIF manifests or HTML displays to render in no time. We also push additionals helped data into these templates, like if the user (looking at the output) is logged in, is admin, the IIIF servers that are configured for to generate IIIF Images and even the Names/Titles of every Webform Element used to ingest/edit the metadata (if one decided to use the same “Labels” as during ingest). Users can create New Metadata Display entities and use them in Viewers/Displays/Search/Views/Maps and during Ingest via AMI. Templates can also be used to “transform data” and then index the transformed data into Solr. basically infinite options.

- Exposed Metadata Display Entities

Metadata Display Entities can also be exposed as endpoints (imagine dynamic Datastreams) on each ADO. for an ADO that lives in your.site.com/ado/uuid you can give a certain Metadata Display an external Name (e.g mods) and it will be exposed at your.site.com/ado/uuid/metadata/mods/default.xml (default can be really any name you want). This is a way of allowing other systems to use your IIIF Manifests, allow harvesting/downloading of your MODS 3.7, Schema.org, GeoJSON, DC, QDC, Bibframe, EAD2002, EAD3, etc. There is no limit on how many you can have and any change on a template will immediately reflect a cache cleanup. The exposed system is very performant and does Cache bubbling to keep data always fresh but still cached. Also, it uses the “exposed and desired” output of each Metadata Display to negotiate the Content-type header and only allows the correct extensions on the endpoint (no way you can ask for a .json if the output is XML). We are working on giving these Endpoints extra API capabilities (REST/SWORD) so you can also configure and expose controlled arguments (an API builder)

- WebAnnotations:

  WE expose a full API for WebAnnotations. With add/edit/update capabilities and temp storage that persists when you log out and come back in your session. These API/endpoints also keep track of which Images (referenced in the SBF) were annotated and create inside the JSON/SBF the W3C valid entries.

- Lazy Image Loading:
  - Twig templates can provide image tags with a special CSS class. If that is the case Archipelago will only request them from the IIIF backend when they enter the View field of the user (+100px)

- Direct access to Files via custom IIIF (mimic) URL:

  Drupal is bad at exposing non Public Files to the world. We wrap this functionality in these modules allowing any file to be downloaded directly if the ADO allows it in it's access rules. Also provided is a special file streaming wrapper that allows remote (e.g S3) files to be seeked and be requested by byte ranges. This allows huge GByte size files (e.g WACZ) to expose it's index first (in the last 64K bytes) and then stream internals on demand in real time, video streaming, etc.This is also very memory performant and bytes and ranges are copied and proxied between remote and requester without consuming large buffers (allowing really large requests to be made)

- View Mode to ADO type Mapper:

  Since Archipelago really does not need multiple Content Types (a single Digital Object type can describe a full range of needs of a repository) but we want that users can decide how each Object is displayed based on the “metadata/JSON” type definition, we provide a Drupal View Mode Mapper. Via this you can decide that an ADO that has the “type” : “Article” JSON value uses a certain View Mode configured with the Mirador Viewer as main Viewer and a special Object Description Metadata Display that shows citations but a “type”: “Book” uses a IABookreader. All this without forcing/setting per Object a View Mode. Still, you can also, per ADO, at will, select a special View Mode that will override this customization.

## AMI: v0.4.0

AMI is our UI/UX batch ADO ingest/edit module. It provides Tabulated data ingest for ADOs with customizable input plugins. Each Spreadsheet (or Google Spreadsheet) goes through a Configuration Multi Step setup and generates at the end an AMI Set. AMI Sets then

can be enqueued or directly ingested, its generated Objects purged and reingested again, it's source data (generated and enriched with UUIDS) CSV replaced, improved and uploaded again and ingested.

The latest version of AMI also includes a Solr Importer plugin that can be used to create AMI ingests and migrating content from existing Solr-sourcable digital repositories (such as Islandora 7); a Linked Data Reconciliation tool that can be used to enrich your metadata with Linked Data (LoD); and new update modes including complete Update, Replace and Append.

- AMI Setup Steps:

  AMI has Ingest, Update and Patch capabilities. AMI has a plugin system to fetch data. The data can come from multiple sources and right now (RC2) CSV/EXCEL or Google Spreadsheets are the ones enabled. Direct from Solr was made available in RC3, and direct from OAI/API is in the works too for future releases. AMI can read files locally from the server, remotely from URLs or remotely from Private Backend Storage (S3). It does parent/children validation, makes sure that parents are ingested first, cleans broken relationships, allows arbitrary multi relations to be generated in a single ROW (*ismemberof*, *partOf*, etc) pointing to other rows or existing ADOs (via UUIDs) and can process rows directly as JSON or preprocessed via a Metadata Display entity (twig template) capable of producing JSON output. These templates can be configured by “type”, Articles v/s 3DModel can have different ones. Even which columns contain Files can be configured at that level.

- AMI Set Entity:

  Ami Sets are special custom entities that hold an Ingest Strategy generated via the previous **Setup steps** (as JSON with all it's settings), a CSV with data imported from the original source (with UUIDs prepopulated if they were not provided by the user). These AMI sets are simpler and faster than “batch sets” because they do not have a single entry per Object to be ingested. All data lives in a CSV. This means the CSV of an AMI set can be corrected and reuploaded. Users can then Process a Set either putting the to be ingested ADOs in the queue and let Hydroponics Service do the rest or directly via Batch on the UI. ADOs generated by a set can also be purged from there. These sets can also be created manually if needed, and any of the chosen settings modified anytime. Which AMI set generated the Ingest is also tracked in a newly created ADO’s JSON and any other extra data (or fixed data e.g common Rights statements, or LoD) can be provided by a Twig Template. Ingest is amazingly fast. We monitored Ingest with Remote URL(islandora Datastreams) files of 15Mbytes average at a speed of **2 seconds per Object** (including all post processing) continuously for a set of 100+.

- Linked Data Reconciliation :

  Using this tool, you can map values from your topical/subject metadata elements to your preferred LoD vocabulary source. These mappings can then be transformed via a corresponding Metadata Display (Twig) template to process the values into JSON-formatted metadata for your specified AMI set.

- Update Modes :

  Normal Update functions as before (completely replace all keys and values with the new source data CSV). The Replace mode will replace only new keys provided/generated by the set. The Append mode (to be used with caution) will add values to existing one and will “array-ify” any destination. The “Keep existing files safe" switch is provided to avoid any Update operations destroying, modifying, messing, or removing existing files—allowing Descriptive Metadata to be updated/changed without affecting precious (and super large) assets.

- Search and Replace:

  This module also provides a simple search/replace text VBO action (handles JSON as text) and a full blown **JSONPATCH** VBO action to batch modify ADOs. The last one is extremely powerful permitting multiple operations at the same time with tests. E.g replace a certain value, add another value, remove another value only if a certain test (e.g “type”:”Article” and “date_of_digital”: “2020-09-09”) matches. If any tests fail the whole operation will be canceled for that ADO. An incomplete “Webform” VBO action is present but not fully functional yet. This one allows you to choose a Webform, a certain element inside that Webform and then find and replace using the same Interface you would see while editing/adding a new ADO via the web form workflow. Should be ready by RC3.

## Strawberry_runners: v0.4.0

This module allows heavy/long running processing on Files/metadata to be configured and generated as a queue item/worker. **Archipelago does not do nor need Derivatives** for Images/Videos/PDFs. All is generated and cached in realtime by the IIIF server. But some larger formats like +Gbyte WARC files or expensive to process HOCR can run via this module in the background without any user interaction in an efficient way.

- Processors:

  Processors are configurable Plugins that can act on ADOs, Files, their Metadata or other Processors. By default only a few Basic Plugins are provided. E.g The

Binary Processor that can execute any command (bash/script/unix command) in your server (in our deployments the PHP-FPM docker container) using files/metadata coming from an ADO as input. Which ADO qualifies can be configured by many parameters. Files can be filtered by “type” and “mime type”, etc. Output of a processor can be also configured: It can be another file, it can be pure data, it can be JSON. Destination can be also configured, the output can be multiple, another chained processor, Solr Index (Strawberry flavor) and File attached back to the Original ADO that was processed, JSON back to the Original ADO. These Processors are also hierarchical (UX) and they can be connected/linked via the UI/UX and exported/imported and Configuration Entities. You can add as many as you want and build complex processing pipelines. Processors always will enqueue (smart) their desired “workers” and then Hydroponics will do the work. Example: The paged processor will count all pages of a given PDF. Will then chain (for each page) with the HOCR processor that will atomically extract a single page of that PDF and output a file that will be indexed into Solr as a Strawberry Flavor. The warc to wacz one is a Binary Plugin that will transform a WARC into a WACZ and reattach the result into the same ADO that provided the file.

- Queue Workers:

  Since each process is enqueued we made sure they are totally atomic and can fail without issues. Their “validity” is evaluated again when the turn for actual processing comes and each one, depending on its destination has failsafe options to avoid overwriting previous ones or running again duplicated. Queue workers are in charge of this and provide all the logic.

## General Code Base statistics

strawberryfield/strawberryfield

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
```

| Language | files | blank | comment | code |
|---|---|---|---|---|
| PHP | 90 | 1761 | 5617 | 9474 |
| YAML | 12 | 34 | 10 | 632 |
| XML | 5 | 0 | 0 | 218 |
| JavaScript | 1 | 34 | 11 | 191 |
| JSON | 1 | 0 | 0 | 38 |
| Markdown | 1 | 12 | 0 | 15 |
| SUM: | 110 | 1841 | 5638 | 10568 |

strawberryfield/format_strawberryfield

| Language | files | blank | comment | code |
|---|---|---|---|---|
| XML | 8 | 33 | 25087 | 129974 |
| PHP | 60 | 1277 | 3918 | 9104 |
| JavaScript | 18 | 386 | 440 | 2430 |
| YAML | 15 | 69 | 32 | 1358 |
| CSS | 5 | 20 | 6 | 138 |
| SVG | 3 | 0 | 0 | 42 |
| JSON | 1 | 0 | 0 | 24 |
| Markdown | 1 | 15 | 0 | 20 |
| Twig | 2 | 0 | 16 | 6 |
| SUM: | 113 | 1800 | 29499 | 143096 |

strawberryfield/webform_strawberryfield

| Language | files | blank | comment | code |
|---|---|---|---|---|
| PHP | 49 | 1466 | 3470 | 8041 |
| XML | 6 | 0 | 0 | 1652 |
| JavaScript | 5 | 78 | 150 | 336 |
| YAML | 6 | 2 | 2 | 169 |
| Twig | 4 | 1 | 64 | 62 |
| CSS | 1 | 4 | 0 | 36 |
| JSON | 1 | 0 | 0 | 25 |
| Markdown | 1 | 18 | 0 | 24 |
| SUM: | 73 | 1569 | 3686 | 10345 |

archipelago/ami

| Language | files | blank | comment | code |
|---|---|---|---|---|
| PHP | 44 | 1168 | 3649 | 8402 |
| XML | 5 | 0 | 0 | 381 |
| YAML | 8 | 24 | 4 | 249 |
| JSON | 1 | 0 | 0 | 39 |
| Markdown | 1 | 0 | 0 | 2 |
| SUM: | 59 | 1192 | 3653 | 9073 |

strawberryfield/strawberry_runners

| Language | files | blank | comment | code |
|---|---|---|---|---|
| PHP | 27 | 585 | 1712 | 3190 |
| YAML | 8 | 5 | 2 | 342 |
| XML | 5 | 0 | 0 | 79 |
| JSON | 1 | 0 | 0 | 33 |
| Markdown | 1 | 15 | 0 | 19 |
| JavaScript | 1 | 3 | 0 | 13 |
| SUM: | 43 | 608 | 1714 | 3676 |