INK 1.0 - Working Draft

Charlie Ablett, INK lead developer / architect
10 July 2017

This document describes current state of INK as of version 1.0 and describes the second stage of our strategy.

Many thanks to everyone who has contributed to this document - Adam Hyde, Kristen Ratan, Jure Triglav, Dr Craig.

INK inception

Adam Hyde got the idea for INK in 2015. In February 2016, we put our heads together to discuss the idea, the challenges, identify some basic use cases and collaborate on an initial architecture.

Development of a proof-of-concept prototype began shortly afterwards and the initial architecture proved viable. Full-time development of INK 1.0 started in September 2016.

Early objectives included developing a framework with the following features::

- HTTP service API
- Basic web client
- Reusable file conversion steps
- Ability to assemble arbitrary steps into a pipeline (recipe)
- Results of steps can be accessed through a user interface
- Develop a step-author community, and enable the community to share steps and recipes easily

Initially INK development focused on file conversions, which satisfies a simple but useful use case. Once the capabilities of the framework were demonstrated, other Coko products - Pubsweet and Editoria - began to use INK for file conversion in early 2017.

INK development has since broadened considerably to include other use cases.

What INK does

INK is intended for the automation of a lot of publishing tasks from file conversion, through to entity extraction, format validation, enrichment and more.

INK does this by enabling publishing staff to set up and manage these processes through an easy to use web interface and leveraging shared open source converters, validators, extractors etc. In the INK world these individual converters/validators (etc) are called ‘steps’. Steps can be chained together to form a ‘recipe’.

Documents can be run through steps and recipes either manually, or by connecting INK to a platform (for example, a Manuscript Submission System). In the later case files can be sent to INK from the platform, processed by INK automatically, and sent back to the original platform without the user doing anything but (perhaps) pushing a button to initiate the process.

To illustrate this consider the case of a Microsoft Word file being submitted by an author to a Manuscript Submission System (MSS). If the MSS is connected to INK, the Word file could be sent to INK and a number of operations performed on that file automatically. For example, INK might convert the file to PDF and HTML, it might identify the abstract, methods, and results sections of the document and annotate them accordingly, and finally INK may identify the research subject area against the publishers taxonomy and return all of this information back to the MSS in the form of converted files and metadata. All this without the author noticing except, if the MSS is nicely integrated with INK, perhaps some of their submission information (eg subject matter taxonomy) has already been automatically selected. This is just one simple example of how INK can help publishers. There are many more examples and ideas and we are working hard to meet these needs with INK!

For the more technically minded, INK is a standalone Ruby on Rails application with a HTTP API for receiving files and data and executing customizable actions on them. INK steps (added as Ruby gems) can be created and assembled into larger recipe pipelines through an intuitive user interface or an API. These steps can execute custom code in multiple languages, execute system commands, or hit other APIs, and the output of each step in a recipe can be easily inspected for debugging. Any set of steps and recipes can be built or customized by any organization, and its modularity and extensibility allow it to support a huge array of use cases.

INK is designed to be a community tool and anyone can build a step or construct a recipe. As the community adopts INK, there will be a broad step library that will lower the barrier to new uses of INK over time.

INK terminology

What INK does is simple to understand as described above, but how INK technically achieves this is pretty complex. To further understand how INK works, let's first take a closer look at the INK lexicon.

INK
Ingest 'n' Konvert. An ETL (Extract, Transform, Load) tool developed by the Collaborative Knowledge Foundation. It's the software you're reading about right now.

INK Instance
A copy of INK software running on a server. Many facets of an instance can be fully customized to meet the needs of different organizations and groups.

API
Application Program Interface. An interface provided by a software system for other software systems ("**consumers**") to communicate with. Generally not meant for humans to use directly. Humans use a client - itself a consumer - with a User Interface that translates user activities into API calls.

API Consumer
Software that communicates with an API. Generally, this means that when the API Consumer makes a request to the API, the API fulfills it and sends back payload. An example of an API Consumer would be Editoria, xpub, or PubSweet.

Client
A user interface meant for use by publishing production staff.

Rails
Ruby on Rails is a software framework for creating Web applications. **Gems** - external modular libraries written in Ruby - can be loaded into Rails in any combination. INK leverages this flexibility to allow a custom combination of gems in an INK instance.

Step Class
*Sometimes we refer to a Step Class as just a **step,** or **INK step***
These are the reusable processes that perform operations on files or content. They're written in the programming language Ruby.

Examples may include:
- "Translate an epub file from Vietnamese to Norwegian using an external API"
- "Extract all the images in this document and make them all greyscale"
- "Convert an HTML file into a PDF using Vivliostyle Electron"

Step Gem

INK defines a Step Gem as a gem with a particular structure, which contains one or more **Step Classes** in it. When a Step Gem is installed on an INK instance, the Step Classes become available for users to use. Step Gems can be stored and run from INK locally, or hosted on remote version control (e.g. Gitlab).

Recipe
A file/content conversion pipeline. A recipe consists of one or more **Recipe Steps** in a defined order.

Recipe Step
As part of a Recipe (a template), it knows its position and what *Step Class* supposed to run when the time comes to execute.

Process Chain
When a Recipe gets executed, a Process Chain is created (with as many Process Steps as needed). A Process Chain knows when it's been executed, the input files it's been given, and information that results from the *Step Classes* being run in the order specified by the *Recipe*.

Process Step
A Process Step is part of a *Process Chain*. Process Steps are copied from the template *Recipe Step* upon execution. It stores the result of its associated Step Class execution.

Account
An entity representing a user or organisation in the INK system. An Account has login credentials.

Admin
A type of Account with broader privileges and access than a non-admin Account. An Admin Account may or may not be a *system administrator* so this distinction is made.

System Administrator
An IT person who has access to the server that INK is hosted on and manages it. They would set up an INK instance, install step gems when needed, and monitor system processes. System administrators may or may not have an INK Account.

Service
An Account may have up to one Service associated with it. A Service has a token, which it can provide to the API for authentication.

INK 1.0 Features

INK has the following features as of 1.0:

Recipe management

- An Account can create a recipe or modify an existing one, specifying its name, description and Step Classes in order.
- Recipes can be set as public or private. A private recipe can only be viewed and executed by the account that created it. A public recipe can be viewed and executed by anyone.
- An Account can set parameters for individual Recipe Steps.
- A Recipe can be archived if it's no longer needed - past executions still be viewed but not executed.

Recipe execution

- An Account or Service can execute a recipe with provided file/s, and parameters. Size and number of files is limited by web server settings (e.g. nginx max file size) and system resources.
- Parameters are used to modify Step Class behaviour. One-off parameters provided at execution time for a Recipe Step override parameters with the same name specified in the Recipe Step.
- Step Classes can set parameters as required. If a required parameter is missing, execution halts.
- An Account or Service can retry an execution without having to re-upload the file/s - INK will use the existing input file/s and parameters they originally provided.
- Execution requests can include a callback location. INK API will send the payload to that callback location once the asynchronous execution process has finished.
- An Account or Service can see their past executed process chains (pipelines) with results for each Process Step included - errors, notes, information (e.g. step gem version, Step Class), and whether or not the Recipe Step considers the execution to have been successful (which is generally up to the Recipe Step itself, usually set to "true" unless an error was encountered).
- An Account or Service can access a log that is useful for debugging Process Steps that have not performed as expected.
- An Account or Service can download files associated with their own past executed Process Chains. This includes input files for the Process Chain itself (what file/s they originally provided) and output files for each Process Step.
- An Account or Service can subscribe to process chain execution events to provide real-time updates. Execution events are sent via the Pusher protocol to subscribed listeners.

Authentication and accounts

- An Account can log in and perform subsequent requests with a server-provided JSON Web Token (JWT) payload for authentication.

- An Account can have a Service associated, which can be given a token for it to use rather than credentials. This is intended to allow non-user services (such as Editoria) access to INK without the use of credentials.
- Authorisation: Accounts cannot access each other's process chains, nor the files associated with other accounts' process chains. A Service has access to the Account's assets for purposes of authorisation.
- An admin Account can see a list of accounts and services.

Server administration

- An Account can be have the role Admin, which allows access to specific information.
- A system administrator can customise which step gems INK loads and makes available for execution.
- An admin Account can see a list of these Step Classes, listed by gem, that INK has loaded.
- An admin Account can see the status of essential components used by INK (e.g. execution engine) to quickly determine if any of them need attention.
- An admin Account can monitor the execution engine to check for fatal errors or failed process chain executions.

Step authorship

- Step Classes contain custom behavior.
- Step Classes can include default parameters (which can be overridden by parameters supplied as part of a Recipe, or at runtime).
- Basic support methods are available for Step Class authors to use.
- All Step Classes subclass the InkStep::Base class (found in the ink-step gem) which provides methods such as **`working_directory`**.

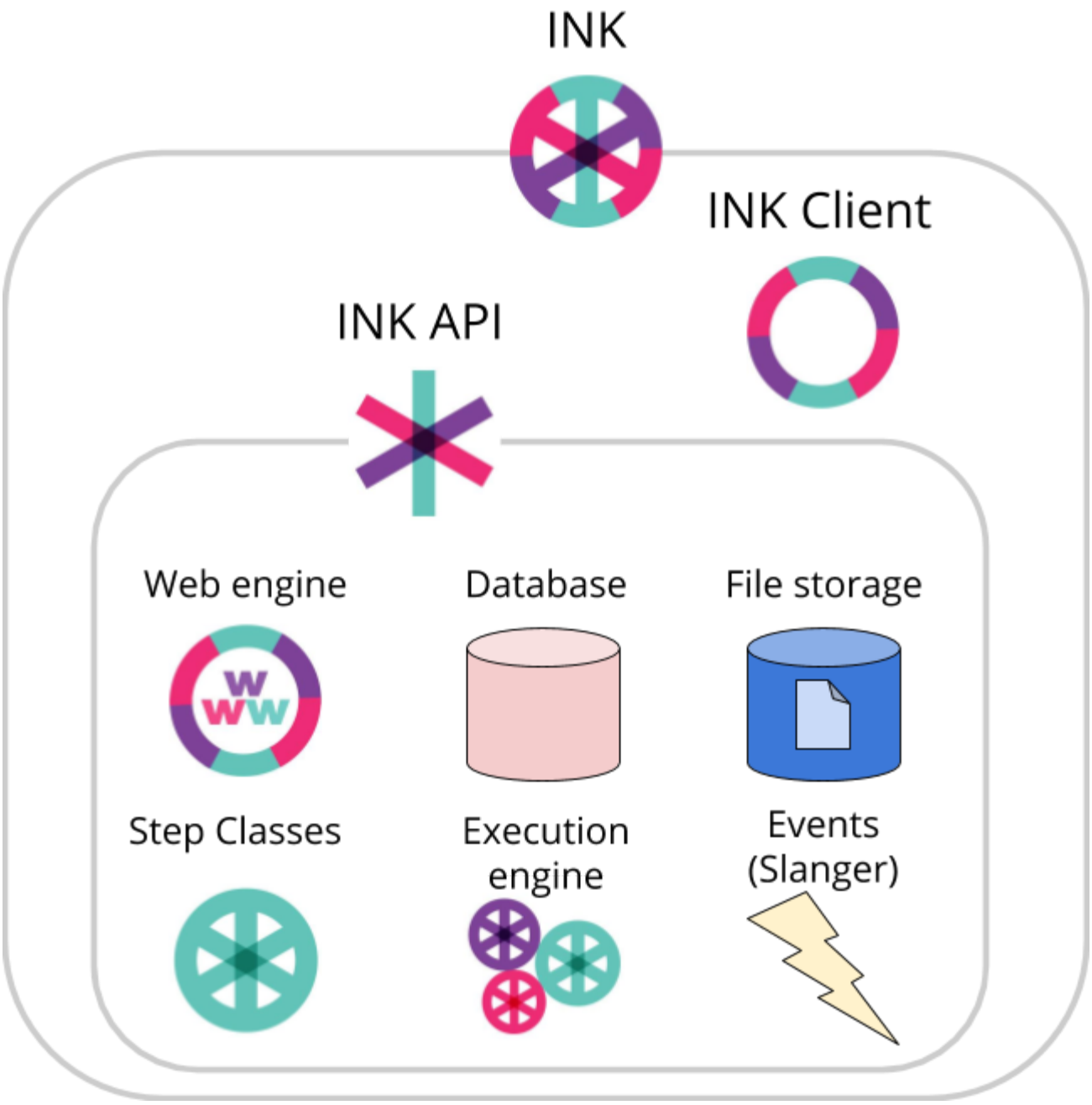
INK API is thoroughly tested with unit, controller and integration tests using the Rspec framework.

How INK works

The following pages detail the INK architecture and how it works.

Architecture

INK is composed of several parts as illustrated below. Each of these elements is described in detail throughout the rest of this document.



INK API

The INK API is the endpoint for third-party services. It's not meant for users to use directly (that would entail very long **`curl`** commands!). It's more likely that the user is logged into Editoria or PubSweet and that service uses INK behind the scenes. Most users who use those third-party services won't realize they are using INK at all.

INK Client

Users, such as publishing production staff, who want to interact with INK directly use the INK Client user interface. The INK Client is written in React.js, using Redux. It's built with Webpack. INK Client calls the INK API and is useful for both demonstrating INK API features and as a basic administrative interface.

Using an event subscription model, live updates are sent from the API to the client for real-time updates on INK processes.

INK API components

INK API comprises of several parts. Let's break these down and take a closer look at each.

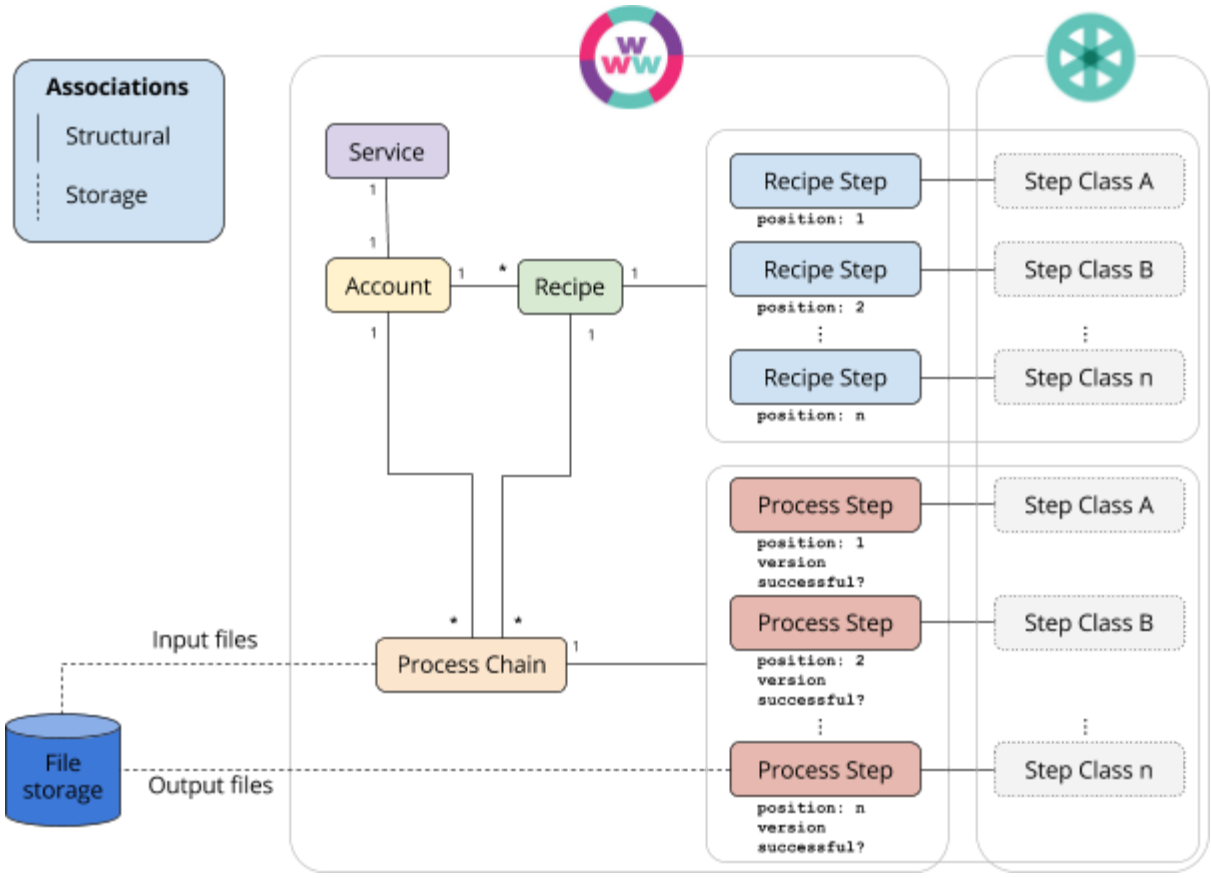
INK Web

INK Web is written in Ruby using Rails 5 as an API with a simple object model and no views nor assets. It responds to and delegates API requests, manages the connection to the database, persists objects and retrieves files when users wish to download them. Devise is used for authentication, which is done via JSON Web Tokens (JWT).

File storage is decoupled from INK Web and can be offloaded to a separate server if desired (e.g. S3).

INK web architecture

The object model for INK Web is indicated below. The objects with solid borders in this diagram are persisted to the database. The dashed objects are not. Structural associations are indicated with solid lines. Multiplicity is shown between objects: one-to-one and one-to-many.



Available steps

INK Web dynamically finds Step Classes available upon Recipe creation or execution. The Step Class must fulfill the following criteria to be dynamically loaded:

- It's included in a gem which is installed on the INK instance (included in the installed bundle when the Rails server started)
- Its name ends with **Step**
- The gem has a structure that INK Web expects and is in the expected directory
- It's a subclass of **`InkStep::Base`**

Events (Slanger)

INK API uses Slanger (open source event service that uses the Pusher protocol) for handling events. API consumers subscribe to Slanger channels for real time updates of Recipe execution. As the asynchronous process progresses, and the INK Client updates the user in real time.

The events are:

```
Process Chain started

Process Step 1 started
Process Step 1 completed

Process Step 2 started
Process Step 2 completed
...
Process Step n started
Process Step n completed

Process Chain completed
```

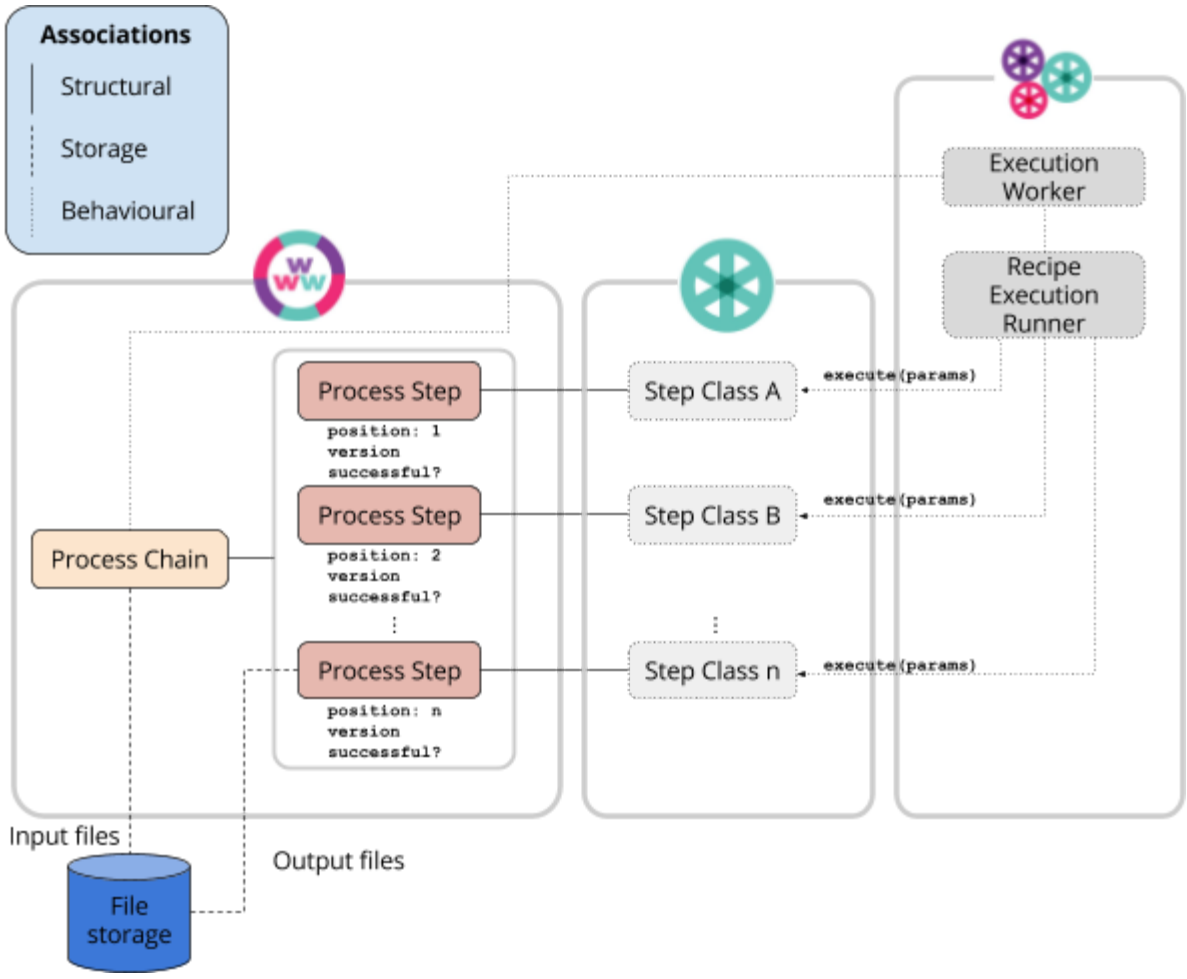
Execution Engine

The INK Execution Engine uses Sidekiq's multithreaded Worker asynchronous processes to do the heavy lifting. The Execution Engine can be hosted on a separate dedicated asynchronous job server if required.

When an INK Recipe is executed, the Execution Engine starts an asynchronous worker process (**`ExecutionWorker`**). The worker process starts a **`RecipeExecutionRunner`** that iterates through each Step (Step Class). Each Step is given its own working directory and any parameters sent by the request. Then, it executes the code inside the **`perform_step`** method.

Execution Engine architecture

The following shows an abridged object model of INK Web as it interfaces with the Step Classes and Execution Engine.



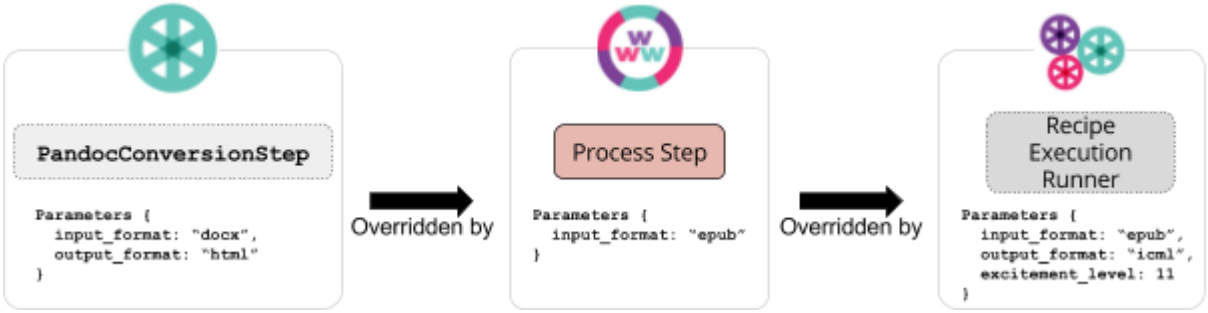
Combining parameters

Parameters can be specified at one of three levels.

Step Class level
A Step Class author may specify some default parameter. Unless it's overridden, the step will use that default.

Recipe Step level
A Recipe author may specify some default parameters for each Recipe Step. These parameters may override any of the same name from the Step Class level.

Execution level (runtime)
When an execution call is made to the API, parameters can be included. These parameters will override any of the same name from the Recipe Step and may override on the Step Class level (it depends on how the Step Class handles them).



The above illustrates an example of parameter overriding.

- **`PandocConversionStep`** is shown as defining two parameter values for **`input_format`** and **`output_format`** (these lines).
- The **`input_format`** value will be overridden by the value given to the Process Step (**`"epub"`**). Since the **`output_format`** value isn't there, **`PandocConversionStep`**'s value for **`output_format`**, **`"icml"`**, remains unchanged.
- As we move on to RecipeExecutionRunner's runtime execution parameters, the **`input_format`**'s value of **`"epub"`** is kept (as it's identical) and the **`output_format`** value **`"icml"`** overrides. As **`excitement_level`** is new, it's added. For extra excitement.

In the example shown above, the final parameter set used by the **`PandocConversionStep`** for this Process Step at runtime would be:

```
Parameters {
  input_format: "epub",
  output_format: "icml",
  excitement_level: 11
}
```

Step Classes

A step class defines some sort of behavior that is carried out against file/s or content. Step Classes can be written by anyone and can be hosted locally, on a git instance or on rubygems.org.

Each Step Class has a *version* which is set either at the gem or step level.

Below are a sample of the Step Classes written by CoKo:

`InkStep::PandocConversionStep`
Takes parameters **`input_format`** and **`output_format`** to execute a Pandoc conversion (**`InkStep::PandocDocxToHtmlStep`** subclasses it and overrides the parameters on this line).

`InkStep::VivliostyleHtmlToPDFStep`
Converts a HTML file to PDF (no parameters)

`InkStep::XsweetPipeline::DownloadAndExecuteXslViaSaxon`
The Editoria pipeline steps are all subclassed from this one. The step author specifies a URL, the step downloads the XSL file from that location and applies it to a HTML file using Saxon.

`InkStep::RotThirteenStep`
A demonstration step that applies a basic cypher to text

`InkStep::ShoutifierStep`
A demonstration step that transforms all text in a plaintext document to uppercase. It accepts a parameter called **`punctuation`** - if set, any full stops (periods) are set to that value. If not, it defaults to **`!!!`**

Here's a selection of what step classes can do, with some examples of each:

File conversion
- docx => html
- epub => pdf

Validation
- Is this HTML 5 document valid?
- Detect grammar errors

Enrichment
- Adding contextual annotation

Entity Extraction
- Extracting all the images in a document
- Compiling a list of places in a document and mapping them

Content Transformation
    Translation
Metadata management
    Reading metadata
    Adding authors to metadata
Analysis
    Detecting widows and orphans
    Characterisation via JHOVE
Distribution
    Uploading to another server
    Publishing to WordPress

There are many different things that a step could do. Here are some samples, and as our partners and community brings more use cases, the selection of available step classes will keep growing.

INK step classes are designed to be file- and content-centric (which could come from a URL, for example). Examples of step class behaviors which INK steps are not really designed to handle:

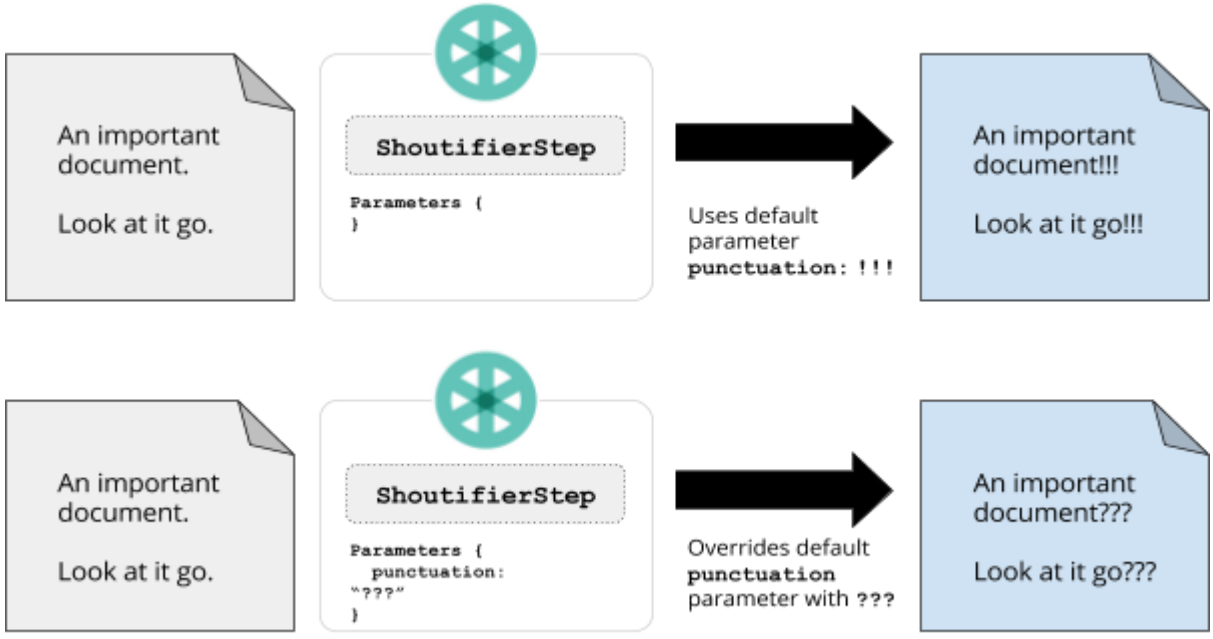
- Notifications - e.g. Mattermost, HipChat, SMS - should be handled by the API consumer
- User interaction (terminal or via graphical interface)
- Excessive use of system resources
- System calls that could potentially destabilise the server (e.g. **`sudo shutdown now`**)

Step authors are free to write a step that does one of the above, but we don't recommend it.

It's also important to note that Step Classes can comprise of any code the Step authors wish to write. Generally, we imagine each Step to do 1 thing and do it well as this makes Steps clean and neat and expands the possibilities for reuse. However, there may be any amount of complexity and/or conditional logic in any Step Class.

Parameter handling

A step class can do anything to a file. The figure below describes **`InkStep::ShoutifierStep`** in action, feeding the same file into the same step with different parameters:

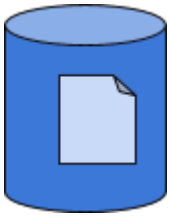


JSON manifest

When a Step Class has finished, a JSON file manifest is collected from the contents of the Process Step's working directory and made available to others. It details which files are in the directory and generates a checksum for each. A semantic tag is added to each, **`:new`**, **`:identical`** or **`:modified`**. This information can provide a longitudinal overview of the processing done by the entire Recipe.

```
[
  {
    path: "important_document.html",
    size: "2.8 kB",
    checksum: "111017ee94636df62daa98257cc1bfeb",
    tag: :new
  }
]
```

File storage

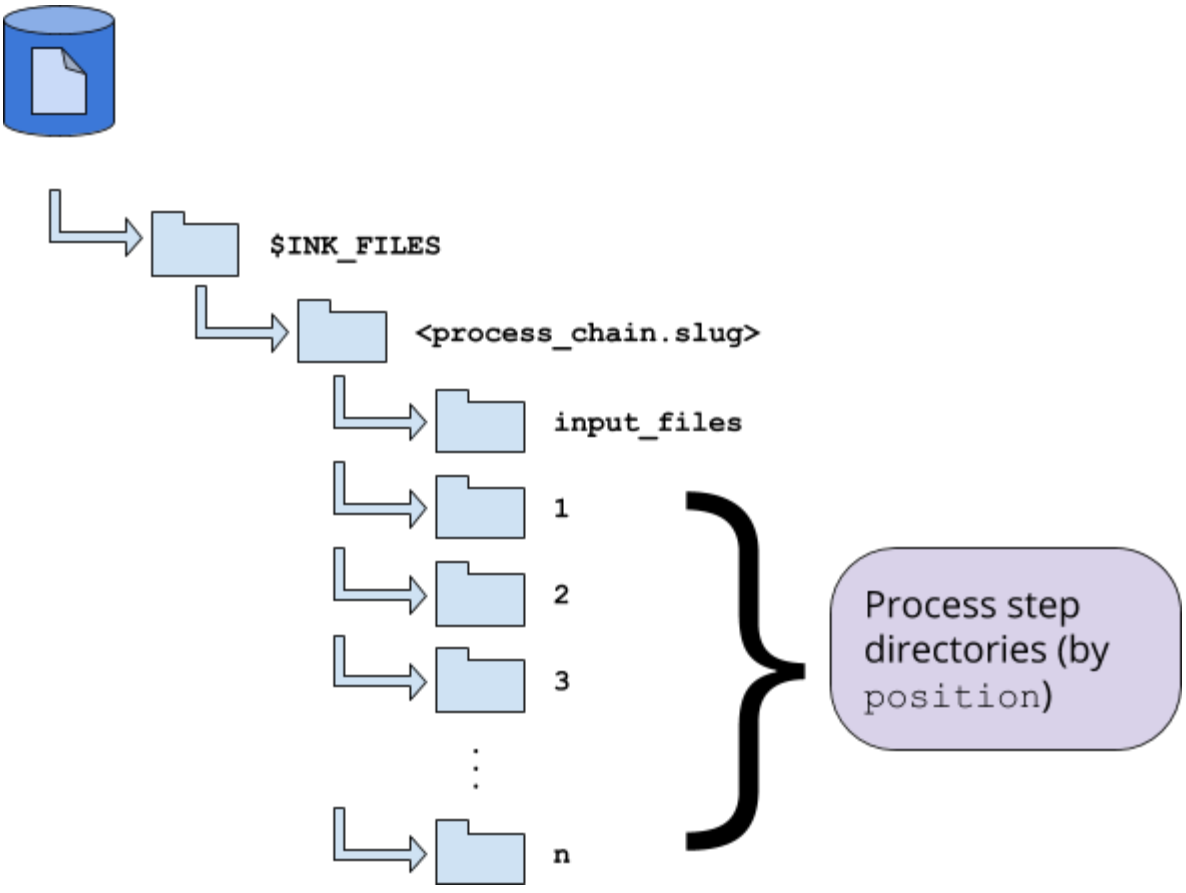


File storage is an integral part of INK. Like a lot of INK, the location is customizable - files can be hosted on the cloud (e.g. S3), another server on the network or the local filesystem.

Directory structure

We use the shorthand **`$INK_FILES`** to reference the location where an INK instance stores its files. It will be different on every INK instance.

When an execution is initiated, INK clones the Recipe into a new Process Chain. The new Process Chain has a unique *slug* (e.g. **`12345abcd`**) that is used to identify its location in the **`$INK_FILES`** directory. A directory called **`input_files`** is created, along with a directory corresponding to the position of each Process Step (1 to *n*). See the directory structure following:



Once the file structure is ready, the files provided by the user are placed into **`input_files`**. Before each Step Class executes, INK copies the files from the previous step directory (or **`input_files`** if it's the first) and the Step Class runs its code against the files in its directory.

This structure means that while there is some duplication of unchanged files, the result of each step is preserved. This is *extremely* useful for posterity and debugging purposes, especially for publishing production staff troubleshooting or improving their recipes or steps.

In addition to the files written by the Step Class, INK also includes a logfile which also be helpful for debugging. It is downloadable by end users.

The future of INK

INK 1.0 presents a solid core. While we could continue adding features to INK ad infinitum, instead (with the exception of Step Modes and some admin tools - both described below) we're going to switch focus to building Steps to perform various tasks for publishers. This is because INK 1.0 is *already* more powerful than we require for most of the work Coko is currently involved with and turning to Step production will help us improve the Step Class developer experience and also understand what features to consider for INK 2.0. We hope there will be others joining us in building a Step library for publishing since INK is most successful when a centerpiece of a thriving community working together to solve shared needs.

Next Steps

Here's what we're considering for the next phase of INK.

More steps!
We aim to cover more use cases and get some more useful steps into the wild. Here are the ones we've got on the immediate horizon:

- Utility steps
    - Zip/tar into an archive (or unzip/untar)
    - Third-party API call
    - Collect all modified files from previous steps
    - Email all files
    - Download a file from a URI
    - SFTP files to external store
    - Generic terminal command
- Generic Steps - These steps represent a wide variety of file conversion tools that can manage multiple type of conversions and validations. These steps will invoke a command and pass on any parameters to fulfil various conversions/validations etc.
    - Calibre
    - Mogrify (batch image processing)
    - Convert (Imagemagik command)
    - HTMLTidy
    - PDFTKF
    - Vivliostyle
    - Pandoc
    - WKHTMLTOPDF
- HTML Validation
- JATS Validation
- Plagiarism checks

- Interpolation of document structure and extraction/annotation of parts (Abstract, methods, results etc)
- Subject matter taxonomy identification
- Image extraction
- DOI Assignment and registration
- EpubCheck (being written by Richard Smith-Unna)
- Natural Language Processing to identify authors, title, institution names, etc in an academic paper
- Identify place names in a document
- Identify people names in a document
- Convert HTML body to JATS
- Create valid JATS metadata
- Merge JATS body and metadata
- Syndicate content to various services
- Push to Continuum

Account management
Allow Admin accounts to create, remove, and modify Account and Services
Allow

Step Class parameter modes
We're exploring a "parameter modes" feature for Step Classes. If a Step Class has parameters, it might be useful for a Step Author to define some presets. For example, a PandocStep might have a parameter mode **`DocxToHtml`** which would automatically mean **`parameters: {input_format: "html", output_format: "docx"}`**. We hope that this will help Step Classes be more usable by non-developers.

INK Recipe/Step Author Community

As INK has reached a point where it's ready for others to contribute step classes and recipes, we'll be focusing on forming a community around creating and sharing Recipes and Steps. We'll also focus on improving the experience of both developers and non-developers, so that more people can contribute to, and benefit from, INK. To this end, here are some upcoming improvements:

Step Class development tools
Enable non-developers to participate in creating INK steps by providing a utility to write and test Step Class code in the browser - no installations required.

Step workshops
Coko plans to host Step authoring workshops.

INK Website
A dedicated web presence for INK which would host documentation, *Step Class Authoring Guide*, and other useful information for getting the most out of INK and writing effective Step Classes.

Open for input

It’s important that INK delivers value and frees up time from time-consuming, automatable tasks. Please let us know if there are any suggested improvements to the framework or existing steps.