API Documentation Plan

Objective

The Digital Farming Platform (DFP) application programming interface (API) offers standardized access to digital farming functionality and data. Since developers discover and use APIs by way of their documentation, the DFP dev portal must supply practical, versatile content that helps users exploit the company's rich data assets to create novel products and services.

Goals of the Portal

- Describe the DFP API's features
- Explain the DFP API's functionality
- Prevent support issues from escalating
- Help users access Bayer's data assets
- Ensure a satisfying developer experience (DX)
- Encourage and inspire innovation

General Info

Project: Comprehensive developer documentation portal for the Digital Farming Platform API, v 1.1.1 (2019-07-22). The spec is encoded using OpenAPI 3.0.2.

URL: The content now available at the FieldView developer site will be replaced with the new Digital Farming Platform dev portal when it's completed.

Schedule: The DFP will be rolled out according to this schedule (2019-2020).

Core DFP API Team:

- Redacted, Principal Engineer (execution)
- Redacted, Sr. Staff Engineer (data modeling)
- Redacted, Principal Engineer (data modeling)
- Redacted, Group Program Manager (management)
- Redacted, Sr. Director of Engineering (quality)
- Melissa Kinsey, Sr. Technical Writer (documentation and DX)
- Redacted, Digital Strategy Manager (UX)
- Redacted, Global Brand and Acquisition Director (global branding and marketing)
- Redacted, [Title?] (enterprise partner engagement)

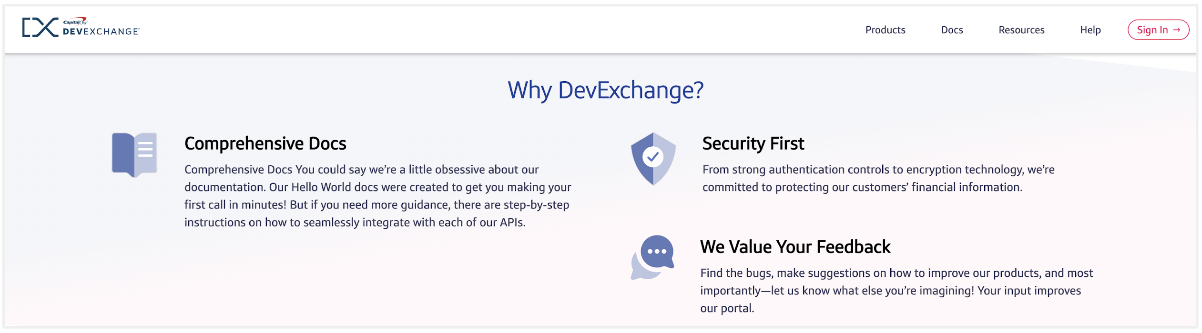
Target Audiences:

The DFP will be useful to several types of clients:

- Bayer engineers
- Internal devs and data scientists
- Platform partners
- Future private-sector enterprise partners
- Public-sector users (for example, the U.S. Department of Agriculture [USDA])

Field Guide User Interface

We propose calling the collective documentation a "Field Guide to the DFP." (For SEO purposes, we can still use the word "portal" in the page metadata and "dev" in the URL.) CapitalOne does something along these lines with its DevExchange portal:



Why DevExchange?

Comprehensive Docs

Comprehensive Docs You could say we're a little obsessive about our documentation. Our Hello World docs were created to get you making your first call in minutes! But if you need more guidance, there are step-by-step instructions on how to seamlessly integrate with each of our APIs.

Security First

From strong authentication controls to encryption technology, we're committed to protecting our customers' financial information.

We Value Your Feedback

Find the bugs, make suggestions on how to improve our products, and most importantly—let us know what else you're imagining! Your input improves our portal.

Although Atom isn't an API, its Flight Manual is conceptually similar.

The DFP portal will have an inviting UI, tidy use cases to point devs toward the right content, solutions to common problems, and plenty of code samples devs can copy and customize for their own needs.

All documentation will conform to the DFP API schemata, conventions, and patterns. Until full design specs are available, we'll use the document specs outlined at go/dfp-doc-style. We'll follow the word style documented at go/list.

Our UI vendor will use the idea of a vintage field guide as the inspiration for font selection, color palette, and graphics. This theme will work well with the agriculture domain.

Of course, the portal UI must be uncluttered, and it must be consistent with Bayer's corporate identity and with the FieldView branding. But it need not be identical, since the dev portal targets a different demographic than the FieldView product.

Developer Experience (DX)

CONTRACT-FIRST APPROACH

The Field Guide to the DFP will offer a peak developer experience, allowing efficient discovery and use of REST [representational state transfer] API resources and functionality. To that end, we're taking a contract-first (a.k.a. "design first") approach to the API. That means framing the API contract before writing the corresponding code. This approach requires more planning and resources but offers several crucial advantages:

- **Product focus.** The contract-first approach treats the API as a product, with the consumer of that product—the developer—at its center.
- **Developer-centric decision making.** A contract-first philosophy shines the spotlight on the concerns of our developers, who in turn are trying to meet the needs of their respective customers.
- **Domain-centric thinking.** A contract-first approach also helps us keep the unique demands of the digital agriculture domain in mind. Seasonal tasks, weather, and perishability are irrelevant to consumers of the Edmunds API or the Stripe API, for example, but they're of paramount importance to our developers and their customers.

USABILITY

In addition to architecting the content intuitively and making it visually appealing, the Field Guide to the DFP must offer the kind of straightforward navigation users expect from any well-designed platform. If the user has to consult the documentation just to get oriented, we've done something wrong.

We must keep the developer in mind at every turn. Usability doesn't mean showing devs how the DFP works—it means showing them how it could work for *them.* A good DX will inspire DFP users to explore, imagine, and invent.

Navigation

The dev portal will have left-sided navigation, with layered accordion menus that allow users to drill down several (probably no more than three) levels. The menu options should be dynamically highlighted—that is, when a user mouses over a menu, the items should change color or move. When selected, an item should respond in some other way. The right side of the portal should be filled with relevant code samples. Users will be able to tab from one language to another to see code samples in their preferred language. These samples will be linked to more verbose code housed in a gist.

Accessibility of the portal documentation must be prioritized from the beginning (see Accessibility).

The dev portal will be equipped with context-sensitive search filters, some of which can be combined, allowing users to narrow their results in several ways. For example, the following filters might be useful:

- By relevance
- By date or season
- By popularity
- By software language (Java, Javascript, Scala, and Python)
- By GPS coordinates or some other geographic filter
- By use case, such as tilling, planting, scouting, harvesting
- Perhaps by other parameters, such as most commented

Accessibility

The portal documentation must be accessible to all users. Not only is it good practice to bake in accessibility, but such features may be required for government adoption of the DFP.

We'll follow the recommendations outlined in the Web Content Accessibility Guidelines 2.1 unless there's a compelling business reason to depart from them. For example, video tutorials will be captioned, and downloadable transcripts will be available. Text will be scalable. Navigational elements will be consistent, but persistent navigation will be optional. Images and other elements will be accompanied by metadata, labels, cues, instructions, tags, or other identifiers. Observing these and similar accessibility guidelines improves SEO and enhance the DX for all users. Please see WebAIM for a helpful accessibility checklist.

Localization

Adaptation of the DFP content to a specific locale or market might include language translation, unit-of-measure conversion, time zone adjustment, or additional formatting options to accommodate special characters, such as the umlaut (ü) or *scharfes S* (ß).

In the DFP spec, properties such as `name` and `description` may be translated. Endpoints that support localization will use the `Accept-Language` header field to specify the language preferred in the response. The header will return translated content for the specified Internet Engineering Task Force (IETF) language tag, e.g., `fr, de, en-GB`.

The language translations available for every entity and instance are accessible using the `/translations` endpoint, which returns a filtered list of existing translations, sorted with the most recent first. Translations can be downloaded in bulk for all languages or for a single language. If the requested translation doesn't exist or if no `Accept-Language` header is provided, the default `en-US` will be returned. See the IETF standards for details.

Metrics

The Product group will track metrics like partner adoption and growth trends, monthly active users, unique consumers, heaviest consumers, and top customers by revenue. Security engineers will monitor calls to prevent unauthorized use by bots or other bad actors. For every endpoint/method, Test Engineering will identify the corresponding service-level agreement (SLA) and ensure compliance with latency restrictions, volume, error threshold, and operating limits.

To gauge success and identify trends in the DX, we will gather data that indicate which documentation is most heavily used, who uses it, how often each user returns, which documentation could be supplemented, which is underutilized, and which should be deprecated:

- **Traffic.** Using real production traffic, Test Eng will run preliminary integration tests, such as regression tests and API contract tests, as well as postdeployment performance checks, to validate availability and assess performance. Metrics above and below the gateway will be monitored. Of particular interest for documentation, for example, is call count. A relatively low number of calls suggests that either developers are quickly finding the info they need, or the API is so unwieldy that devs bounce after just a few transactions. Analysis of other metrics, such as developer satisfaction, can clarify the situation.
- **Developer satisfaction.** When the DFP is deployed, we'll ask willing users to give us feedback on the portal and to rate the DX. This will give us a baseline against which to calibrate our progress.
- **TTHW (Time to Hello World)**. TTHW is a good yardstick by which to measure the ease of onboarding. For a stellar DX, we must give devs all the tools and info they need to get rolling quickly (see the discussion of code samples under Navigation).

Content

API SPEC

The Field Guide will contain use cases, common tasks, and tons of code samples mapped to the DFP API specification, which identifies all API endpoints and associated HTTP methods (create, read, update, delete [CRUD]), resource descriptions, parameters, requests and response.

EXISTING CONTENT

Portions of the documents housed in the shared DFP folder can be repurposed for the Field Guide. This content is classified as follows:

- Governance documents (planning, security, metrics, testing, communication)
- Model documents (conceptual foundations of the API)
- Reference docs (API spec and other technical reference docs)

CORE DIGITAL FARMING PLATFORM (DFP) DOCUMENTS

Governance	
DFP Updates, FY19 & FY20	DFP API Specification Roadmap [DEPRECATED]
DFP API Phased Rollout	DFP Engagement and Commitment Process
Cloud Transition to the DFP	Testing the DFP
Model	
Digital Farming Ontology Initiative	Digital Farming Taxonomy
Orchestration Layer	Event and State Taxonomy
Principal Abstractions	Relationship Taxonomy
Prerequisite API Requirements	Attribute Taxonomy
Unified Metadata Vocabulary	Entity Taxonomy
Reference Types in the DFP	Semantic Mapping—BCS Digital Farming Data Stores
Internationalization Language Model	Guidelines for Names and Definitions
Reference	
Predictions	Location, Location, Location
Prerequisites	Life of the Party
Seeds	Seasons Data Model
It’s About Time	The Measure of Success

NEW CONTENT

Quick-Start Guide

Developer onboarding resources should include at least the following:

- Registration and request for access keys
- Authentication and authorization
- Versioning
- Request and response formats
- Error codes
- Changelog
- Brief how-to's for common tasks
- Endpoints in development (possibly with projected release dates)
- Support info
- TL;DR summaries

Code Samples

Offering lots of handy code samples linked to forkable/cloneable gists is perhaps the most important component of the portal documentation. Code samples must be easy to find and use, so they'll appear in the third column, to the right of the documentation itself (see Navigation).

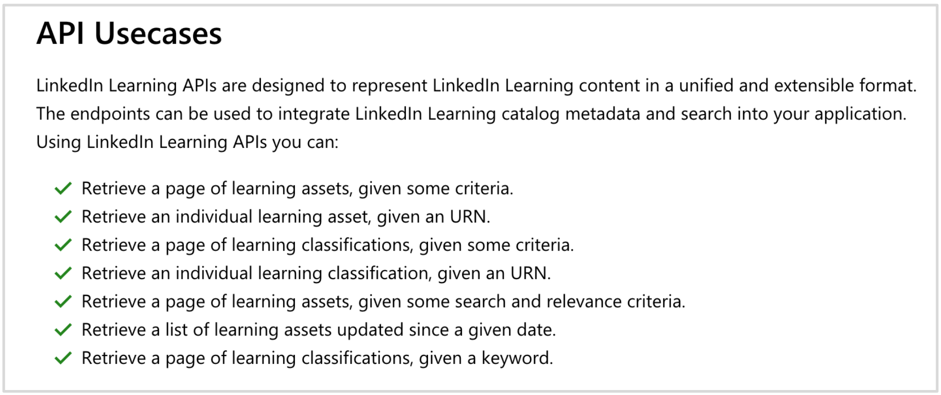
We can generate some of the sample code using Swagger or a similar dynamic autodocumentation tool. This kind of tool will update the code samples whenever the source code changes. However, autodocumented code is just a starting point. To be truly useful, the code must be available on GitHub, annotated, and tied to video how-to's, tutorials, wikis, tools, libraries, and other resources that help developers get up to speed quickly (see TTHW, under Metrics).

Migration Resources

We'll offer migration guides and crosswalks to tie existing endpoints to the new DFP environment. These migration resources are especially important for a good internal DX.

Use Cases

We'll develop a set of concise use cases to help developers find the content they need. Linkedin Learning's dev portal, for instance, shows devs exactly what they can do with the LinkedIn Learning API.



API Usecases

LinkedIn Learning APIs are designed to represent LinkedIn Learning content in a unified and extensible format. The endpoints can be used to integrate LinkedIn Learning catalog metadata and search into your application. Using LinkedIn Learning APIs you can:

- ✓ Retrieve a page of learning assets, given some criteria.
- ✓ Retrieve an individual learning asset, given an URN.
- ✓ Retrieve a page of learning classifications, given some criteria.
- ✓ Retrieve an individual learning classification, given an URN.
- ✓ Retrieve a page of learning assets, given some search and relevance criteria.
- ✓ Retrieve a list of learning assets updated since a given date.
- ✓ Retrieve a page of learning classifications, given a keyword.

SDKs

The DFP API will provide software development kits (SDKs) in at least three languages: Java (which can also be used for Scala), JavaScript, and Python.

Community Features

Developing a vibrant dev community can help promote the DFP to external partners. Community features might include videos, a community discussion forum, a blog, links to Stack Overflow and GitHub, and ties to other social media. We could also generate interest via Meetups, conferences, hackathons, podcast guest slots, and other outreach activities. We should hire a full-time API evangelist f it becomes a priority to promote external use of the DFP.

Sandbox

To encourage exploration during early development, we'll offer DFP-in-a-Box, a stub DFP API. It'll be a standalone service devs can run on their own development machines. This virtual production environment will allow engineers to make mock function calls using test data. They can thereby validate business logic, test-drive a prototype, or just try out the DFP's features and functionality. This playground will give devs a chance to spot glitches and untangle snafus in a simulated ecosystem, eliminating the risk of testing in a live environment.

Making developers' work a bit more fluid and creative encourages community engagement and promotes API adoption and loyalty, which may become increasingly important as external partners discover the DFP and devise imaginative new uses for it. If you'd like to go play, check out PayPal's sandbox, What's App's sandbox, or the beloved Swagger Petstore.

Implementation

API Consumer Feedback

When we have about 30 percent of our documentation finished, we'll recruit a group of non–DFP engineers to try it out. We could shadow them (using contextual inquiry methodology) or use focus groups and interviews to solicit feedback on coverage gaps, content clarity and usability, portal navigability, visual appeal, usefulness of community features, and so on. Then we can integrate the developers' suggestions as we prepare the remaining documentation. We'll solicit commentary from internal devs as well.

Internal Communication

We'll have semiweekly or weekly meetings to discuss portal documentation issues and keep the documentation moving. We're also using go/dfp, #dfp-api, and #tech-writing to communicate.

Delivery

DOCS-AS-CODE APPROACH

Formatting our documentation in Markdown and storing it in a Git repo gives us great flexibility in repurposing and rearranging content, unfettered by proprietary platforms or siloed tools.
A dfp-api-docs directory has been set up within the dfp-api repository. A README file and half a dozen Markdown documents have been added. As of this writing (August 30, 2019), four PRs have been approved and merged, and the remaining existing documents are being converted to Markdown and uploaded to Git. A process has been established for new documents to be converted to Markdown and placed in the docs directory of the dfp-api Git repo (see Document Management for DFP Documents).

VERSION CONTROL

The docs-as-code approach requires document authors to use Git for version control, just as the engineers do. It establishes a trackable review process and lets us incorporate continuous integration into the API documentation workflow. Document authors use the same systems and CICD pipeline the engineers rely on, which will keep the portal documentation in sync with API development.

TECHNICAL WRITING TEAM

Melissa Kinsey, in St. Louis, manages the portal documentation. Other immediate technical writing needs include one or more API spec writers. Two positions have been posted in Seattle.

Each writer must be familiar with the OpenAPI spec and with the tools used to document, test, and iterate on it, such as Swagger, Postman, Atom, and Git. It's an advantage if the writer can read code in one or more programming languages. If not, though, he or she must be familiar with (or be able to look up) differences in terminology and syntax among various languages.

The DFP API will require SDKs in at least three languages (see SDKs). The spec writer will document these SDKs with overviews, release notes, code samples, tutorials, wikis, tools, and libraries.

The writer must go beyond explaining the functionality of individual components. He or she must also document known and anticipated API use cases and illuminate the ways in which the various API components relate to and interact with one another.

Maintenance

We will establish a process for maintaining accurate documentation (see Document Management Policy for DFP Documents), moderating community forums, and generating new documentation as the API expands. We'll shape this process as we go along. For now, users can either handle changes informally, via #dfp-api, or more formally, by submitting a PR.

Questions and comments about this plan are welcome.

Slack: @melissa.kinsey
Email: melissa.kinsey@climate.com

Revision History:		Author(s):	Document Status:
2019.08.13	v0.1	Melissa Kinsey	DRAFT
2019.08.30	v0.2	Melissa Kinsey	Working document