Economic Security Project

MAY 2026

The Future of Tax Filing

Part 2: Building Direct File: Policy and Strategy

Chapter 9: Direct File, fact graph, and a new way of building civic technology products

Jennifer Thomas, Chris Given, Gabriel Zucker

Contents

Summary

- Translating the tax code into a product that asked taxpayers tractable questions and produced accurate returns was a challenging process, which could have been made easier with more and better back-end tools and innovations. Direct File employed two innovations in this regard — (1) the fact graph, a declarative business rules engine

that modeled the tax code and allowed the product to make inferences about incomplete information, and (2) a nascent "taxpert interface" that allowed non-engineers to visualize and interact with the product. We envision expanding on both of these components to develop a new way of building civic technology, which we dub *FactML* in honor of its principal component. (9.1)

- Such back-end innovation would: (1) allow Direct File to be built and scale faster and more reliably, (2) allow states to configure state filing functionality directly in the Direct File product, and (3) open the door for better, more efficient products throughout the civic technology space, including cross-domain linkages. (9.2)
- The impact of these changes can best be envisioned by picturing the process of states building new tax functionality in Direct File, in a FactML world. (9.4.1)
- FactML fundamentally consists of four components: (1) Fact Markup Language (FactML), a generalized XML-based specification for defining legal code as software code, (2) Factual, a portable business rules engine that parses the relationships in FactML, (3) Formative, an opinionated interface library for a user-facing application that collects and asserts information, based on the previous layers, and (4) Formative Studio, a GUI for viewing, annotating, and testing the interfaces and logic contained in the previous layers. Formative Studio is the linchpin that enables effective cross-functional collaboration and makes the entirety of the product tractable in a single interface. Direct File employed functional versions of (1), (2), and (3) (though they each require some documentation and additional abstraction), but only the most nascent version of (4). (9.4.2)
- FactML would solve collaboration and documentation challenges inherent in building such a complex product, it would use this new collaboration potential to unlock state support directly in the product, and it would dramatically speed testing. (9.4.3)
- Taken together, these reforms are a way of working more than they are a single deliverable product. They could be advanced in a variety of contexts, but getting to a FactML world is not a simple linear process. (9.5)

Building Direct File, expanding it, and ensuring its accuracy was difficult. We believe that, with better infrastructure and tooling, it could have been much easier.

On the other hand, across government, teams looked at Direct File and asked, "What did you use to build that?" Other government domains, no less than tax filing, need a process to represent complex eligibility rules, ask questions of users, and create standard representations of this information (e.g., program applications, tax returns) — and Direct File had enviably promising nascent solutions to these problems.

In this chapter, we present the outlines of a robust suite of back-end processes, practices, and products, much of which Direct File had at least begun to build, and the completion of which would have transformed the process of delivering Direct File. The ideas we present here are not entirely novel; many of its components were already implemented in Direct File, and continue to be refined today in successor products. But this chapter elaborates on the

existing work, paints a picture of how these procedural innovations could expand in the future, and outlines some farther-reaching implications of their prospective implementation.

Taken together, this chapter outlines a method of building civic technology, that we dub *FactML development*, in honor of its core component. If fully built out, we believe this way of working would (1) make it easier to restore, expand, and maintain Direct File in the future, (2) open up new possibilities to bring Direct File state filing directly into the product (see Section 4.4.2), and (3) provide a springboard for revitalized government service delivery and the building of new, innovative services across the civic technology landscape.

We base the conclusions in this paper both on our experiences building and maintaining Direct File, and on a round of user research with civic technologists across issue spaces during the summer of 2025.

Sections 9.1 provides more background on the line of thinking that brought us to this proposal. Section 9.2 outlines three distinct categories of benefits the proposal could deliver. Section 9.3 covers foundational user research we did to validate core assumptions underlying this proposal, and flesh out some of the less well developed details. Section 9.4, critically, lays out the substance of the proposal. Section 9.5 discusses how technologists might implement the ideas in this chapter, especially in a world without Direct File.

9.1 Background and Direct File back-end functionality to date

Delivering Direct File required overcoming dozens of challenges, but the challenge of translating the Internal Revenue Code into a humane interface was decidedly the long pole in the tent. Direct File had to ask taxpayers a series of tractable questions, and translate this assorted information about a taxpayer's life into what the taxpayer is eligible to claim. The often fuzzy relationship between these questions and the appropriate tax return is variously described via statutes, regulations, court decisions, instructions, publications, and other guidance. Direct File had to wrangle this problem at a level of accuracy commensurate with speaking with the authority of the IRS.

Some of the difficulty of this work was inevitable, given the historical context. Because the IRS had opted to create the Free File Alliance in 2002 rather than build its own tax filing software, the agency had been sidelined from the work of actually building electronic filing tools, with design and even legal analysis handled by third-party tax preparation companies, rather than in-house at the IRS. Add in the fact that IRS budgets had been shrinking on a per capita basis over the 21st century, and there was a backlog of work and capacity-building that had to be done within the agency to get Direct File off the ground.

But what if, just as we did for taxpayers, we were to treat the people building Direct File as users? We might ask: what would make this work easier? As the scope of Direct File continued to expand, how would we have managed to scale it without being dragged down by the compounding difficulty of maintaining its complexity? What new tools or processes

would have expedited the maintenance and further development of Direct File? And what new possibilities might these tools or processes have made possible?

Put differently, how does a cross-functional team work together to build the core tax logic of a product like Direct File? How do designers and subject matter experts collaborate to develop designs that ask the right questions? How do they work with engineers to get those designs implemented? How does everyone test to ensure it was done right, and that nothing — in the logic, in the designs, or in the implementation — was overlooked? Moreover, how does this work when a team needs to make marginal changes to functionality, either to implement a similar but distinct provision, or due to legal changes over time? How can this entire process be systematized so it isn't just a few people brute-forcing the whole problem?

Direct File incorporated two principal innovative concepts in this vein:

- The "Fact Graph": a declarative business rules engine designed for reasoning about incomplete information, building on prior work dating back to TAXMAN in 1976. The fact graph formally models all the relevant aspects of the tax code, allowing the software to make reasonable inferences about the information it has and needs, without every component having to be manually specified.
    - For example, consider the fact of EITC qualifying child eligibility. Its inputs include, among others, an age test (under 19; or under 24 and a full-time student; or any age and entirely disabled) and a residence test (co-residency with claimant).
    - Suppose the child is 17 but we do not know their co-residency, and we know the taxpayer may yet be eligible for EITC. Correctly specified, the fact graph logic "knows" that we still need to ask about co-residency; but that there is no need to ask about full-time student status. This knowledge can be used to drive the pages a taxpayer sees without manually encoding each of them.
    - Suppose now the taxpayer edits the child's birthday; she is in fact 22. The fact graph logic now knows we need to backtrack and ask about full-time student status.
- The nascent "Taxpert Interface": an interface for and by the tax experts ("taxperts") who were creating the product — that is, the designers, product managers, and subject matter experts whose job it was to ensure that the product faithfully implemented the tax law. The core extant component, the "all-screens" view, is here in the open-source Direct File repo. The taxpert interface functionality was far less built out than the fact graph, as of the end of Direct File in 2025.

These innovations were critical to Direct File's success. But, especially in the case of the Taxpert Interface, they were in their early stages, as the Direct File team had to prioritize taxpayer-facing functionality at the expense of back-end tools. In 2025, year three, a more mature Direct File team planned to for the first time allocate engineering resources specifically to building better tools. This third year was, of course, not to be.

What would it look like if we could pick up where Direct File left off, developing an interconnected web of tools that support the development of Direct File, or other government services? We collectively refer to this web as FactML development — a suite of components that allow products to be built on the fact graph.

9.2 The potential impact of FactML

Before we get into the details of what FactML development could look like, it is helpful to step back and lay out the benefits of investing in this approach in the first place.

9.2.1. Improving and accelerating a revived Direct File

These are the tools the Direct File team wished for and planned to develop; they would have made the product more stable, and made it possible to deliver more functionality more quickly, especially increased tax scope. If these tools were built out, a future decision to bring Direct File back would gain momentum more quickly. These tools would enable the IRS to review the resuscitated product for compliance with current year tax law, make swift updates, and rapidly begin expanding tax scope. They would greatly reduce the risk of accuracy errors occurring in the software and tarnishing Direct File's reputation.

> *Outcomes: Direct File is able to relaunch within weeks of a decision and features a more aggressive roadmap for incorporating new tax scope than would otherwise be possible.*

9.2.2. Better solutions for more states in Direct File

Direct File's 2024-2025 approach to state taxes worked, but, as we argued in Chapter 4: Direct File and state taxes, it was not ideal. Developing and maintaining standalone state tools was expensive for states; the two-part user experience was not good enough for a meaningful minority of taxpayers; and the separation of federal and state products made the prospect of expanding to multi-state returns daunting at best.

We believe FactML development would allow state tax agencies to build out support for state tax returns *within Direct File* (the solution discussed in Section 4.4.2). This would provide taxpayers with a more seamless and intuitive experience, and provide a solution to the partial-year return problem. It would also save significant costs for states, who would not need to host independent products. They would not have to manage account creation or maintenance. They would not need to manage e-filing or reject resolution. They would not need to consume the Direct File API for exports, or route that data to the appropriate points in the state return. Even the creation of state questions would be a much more streamlined process than in the status quo. Overall, the cost to states would fall precipitously.

> *Outcomes: Taxpayers more easily file state tax returns and Direct File expands more quickly to more states.*

9.2.3. Non-tax services, and innovative cross-domain interoperability

Government tech teams working in other domains, just as in taxes, also need a process to represent complex eligibility rules, ask questions of users, and create applications/returns representing this information. Indeed, the Direct File team members had experience with numerous government services and programs, and the team recognized the potential for generalizing these solutions to other problems. The team also noted that the complexity of taxes seemed to be the most challenging of all government systems, meaning that a solution robust enough for tax would very likely work for other domains, too.

Not only would FactML development be a best practice that would make it easier to build benefits applications across government, but its shared framework would make it possible for different government applications to work together. Just as state taxes could build on the fact graph of a federal tax return, state benefits could build on the fact graph too. That is, FactML could schematize the intersections of eligibility rules for different programs, and enable services that allow users to import and reuse information, that recommend other programs you might be eligible for, or that apply for multiple programs in a single step — even if they're administered by different entities.

> *Outcomes: New, innovative government services are delivered at reduced cost and leverage user-controlled data sharing to reduce burden and increase uptake.*

9.3 User research: the problem and the need

This product exploration began with a set of hypotheses, collectively stipulating that something like FactML development would solve real problems for teams building government products.

To test these hypotheses, and explore further what a solution would need in order to be successful, we held a series of structured conversations with technologists who worked on (a) Direct File and associated products, (b) non-Direct File tax products, and (c) non-tax products, primarily in the space of safety net benefits applications. This comprised nine total sessions with eleven total technologists, as well as a full-day retrospective with six members of the Direct File team. We also explored as much as possible the tooling and resources surrounding other government products.

Our core hypotheses were largely borne out by the user research process:

- *Hypothesis #1. We think the task of translating statute/regulations/policies into a product and scaling it is hard.* This was well supported by our research.
- *Hypothesis #2. We think updating digital services in response to rule changes is hard.* This was partially supported by research. While this is clearly true for big and highly complex tools interacting with dynamic rule spaces like Direct File, not all non-Direct File teams felt updates were a significant challenge.
- *Hypothesis #3. We think people who know "the rules" should be directly involved in the creation and maintenance of digital services.* Our research validated that such subject

matter experts are already deeply involved in design and testing, and most want to be empowered to contribute directly.
- *Hypothesis #4. We think non-engineer users will need a "no-code" environment to interact with the details of the product.* Our research largely supported this. Some non-engineer users had worked with "light" tools like YAML, but they still wished for an easier way. Most had not tried to interact directly with the product because the tools were not there or were not practical.
- *Hypothesis #5. We think taxes are the most complex domain for government digital products.* This was supported by our research.
- *Hypothesis #6. We think private tax software companies have their own internal tools to maintain and manage their products, although there is no publicly available tool of this type.* This hypothesis appears to be true, though it was not directly surfaced in our user research sessions.

Exploring the difficulty of building and maintaining a product, we found the following:

- *Lack of the right tool for the task is costly.*
    - Every task takes more time and it means the cost of meaningful iteration is often too high.
    - Most processes are kludgy and manual. Decisions get stuck in e-mail round robins, changes are delivered in spreadsheets, PDFs, Word docs, JIRA, Figma, etc.
    - The smallest update can get stuck in a long backlog if every change depends significantly on engineering.
    - Taken together, all of this pushes all our agile good intentions toward something more like waterfall development. Our build-measure-learn cycles can become perforce superficial.
- *Testing is a huge pain point.*
    - People successfully use the all-screens view, design Murals, tax recipes, and loadable scenarios to test accuracy, but most of those tools are band-aids.
    - It's hard to spot what's missing from a flow. It's easier to see mistakes that are present than to realize a screen or question is missing altogether.
    - It's hard and slow to test different scenarios and permutations in the user experience.
    - Testing happens too late in the process, under time pressure, and it usually takes an expert to create a scenario and recognize if it's correct in the product.
- *People need to understand the system they're building.*
    - Teams need to see the logic behind the product and the screen flows.
    - Teams need to see both the high-level structure and the details, with easy ways to move between them.
    - People need different views and different data depending on what they're doing — design, content, testing, engineering, legal/policy review.

    - People want to see notes, decision logs, and external references tied to screens – without it being manual note maintenance and without creating information overload in the view.
- *Collaboration can't continue to be MacGyver'd.*
    - Teams cobble together their own methods to collaborate and keep track of work.
    - Work is scattered across too many tools.
    - People are using screenshots with instructions typed on top to communicate with engineering, Murals and PDFs to map flows, JIRA/Git/custom systems for tickets, Slack and email for coordination, Word documents for feedback. It's all fragmented and static; there are too many places to look; and the information does not reliably sync up.
- *The to-be tool needs to:*
    - Manage cognitive load — there will be a lot of screens, annotations, scenarios, options, and filters. How well the tool handles that will make or break it.
    - Handle multiple contributor types, groups, permission levels.
    - Support testing — things like scenario builders, building blocks, and AI help.
    - Offer views tailored to different users and tasks.
    - Make it easy to trace flow logic and understand why things work the way they do.
    - Be accessible to internal teams and external collaborators, like states or legal reviewers.
    - Make it easy to:
        - Compose screens and flows.
        - Understand the logic and the statute underneath it.
        - See and test everything — not just the question pages, but also outputs and messaging.
        - Test scenarios and explore different paths "flatly."
        - Recognize status, approval, and readiness at a glance.

9.4 The details of FactML development

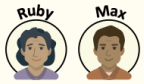
The following is a rough proposal of what robust FactML development might look like.

9.4.1 Storyboard

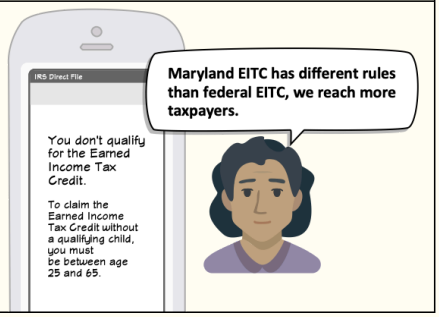
The storyboard below illustrates how a state tax subject matter expert (Ruby) and engineer (Max) might work together to develop new state filing functionality in Direct File, in a FactML world.

Shared tools, Maryland rules

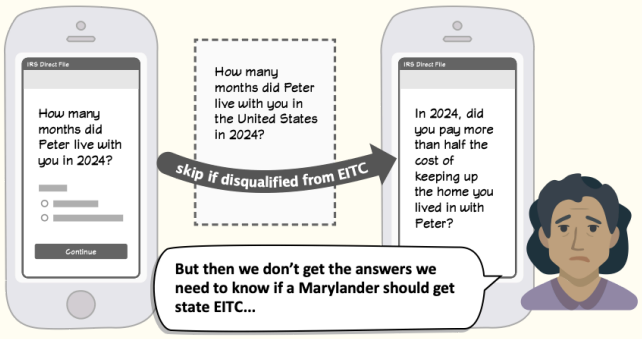
Meet Ruby and Max from the Maryland tax team—Ruby's the tax expert and Max handles engineering



Ruby's goal is to make sure Marylanders who use Direct File can get all the benefits they qualify for—including state benefits like Maryland's Earned Income Tax Credit—without having to bounce between separate tax tools just to complete their return.



Ruby knows Direct File stops asking EITC-related questions as soon as a taxpayer gives an answer that disqualifies them from the federal credit—it's meant to save them from seeing questions that no longer apply.



In the past, handling a tax law difference like this meant Maryland had to maintain its own tool—duplicating federal rules just to apply its state-specific exceptions.

Maintaining such a complex system is hard. When Ruby needed to update the rules to reflect statute changes, she struggled to see how the existing rules were applied—and then wrestled with translating the new rules into tickets that described for Max what needed to change in the tool.

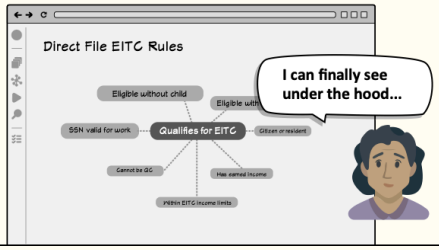
Ruby knew the tax law. Max knew the system. But without a common way to understand and discuss the rules, they kept getting their wires crossed.

But now, Maryland has a place in Direct File.

Ruby opens Formative Studio, the tool behind Direct File, to start customizing the Maryland taxpayer experience.

Ruby pulls up Direct File's EITC rules. In the past, the tools she'd worked on felt like black boxes—she couldn't just *see* their tax rules.

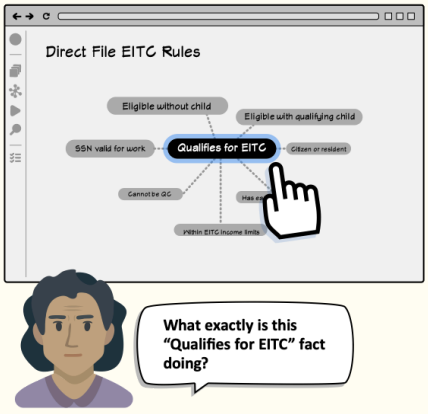
But here, she can—thanks to Factual, which uses AI to translate between the machine-readable "facts" that power Direct File and the human-readable explanations that make sense to experts like Ruby.



Ruby values seeing how the tax logic fits together—but the graph alone doesn't quite match how she thinks about tax rules.

Ruby thinks in terms of eligibility tests and line-by-line instructions—not nodes and connectors.

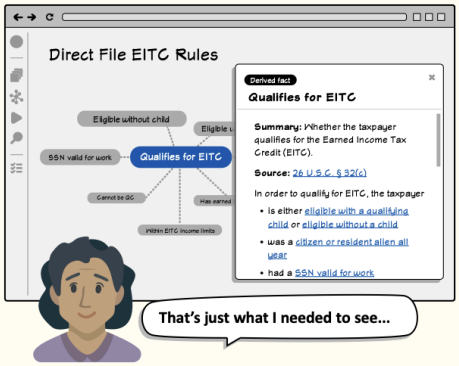
She clicks the fact at the center of the diagram.



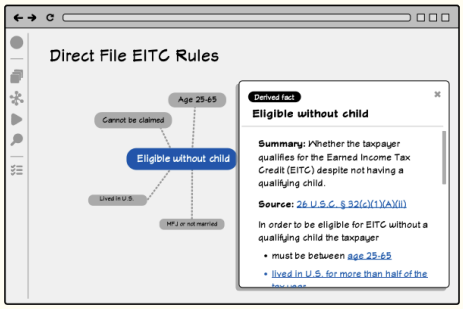
When Ruby brings up the rule, she sees a plain language explanation of the "Qualifies for EITC" fact.

Factual explains what the fact is doing, its "recipe," in language that makes sense to Ruby. This explanation is automatically generated and kept up-to-date using AI.

It even includes a citation pointing to the tax rule in statute.



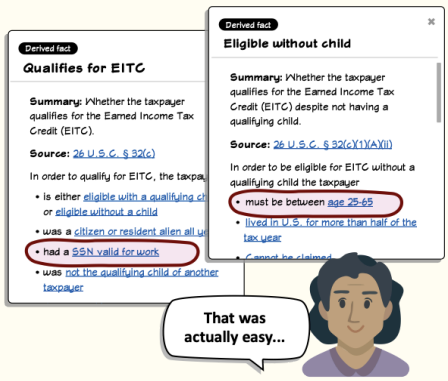
Ruby clicks on links to related facts in the graph, quickly finding another fact she was looking for: the rules for EITC eligibility when the taxpayer has no qualifying child.



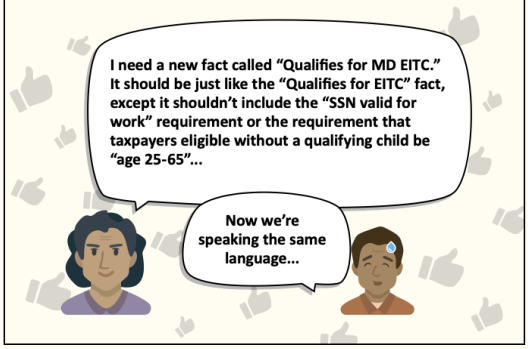
Ruby's not using this tool to look up tax rules—she knows them like the back of her hand.

What the graph and plain language explanations help her do is to *see* where those rules live in the machine-readable version of the tax code.

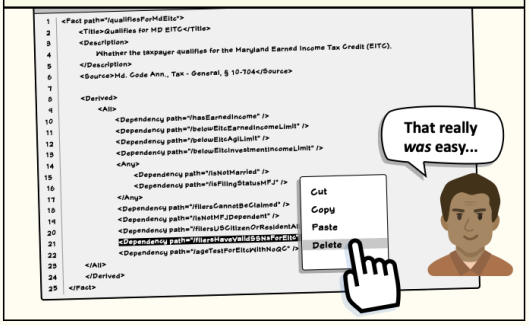
That enables her to quickly pinpoint which facts need to work differently to ensure Marylanders can get the state EITC.



Ruby can now ask for changes she needs with confidence, referencing facts in the product by name. And that makes life easier for Max, too.



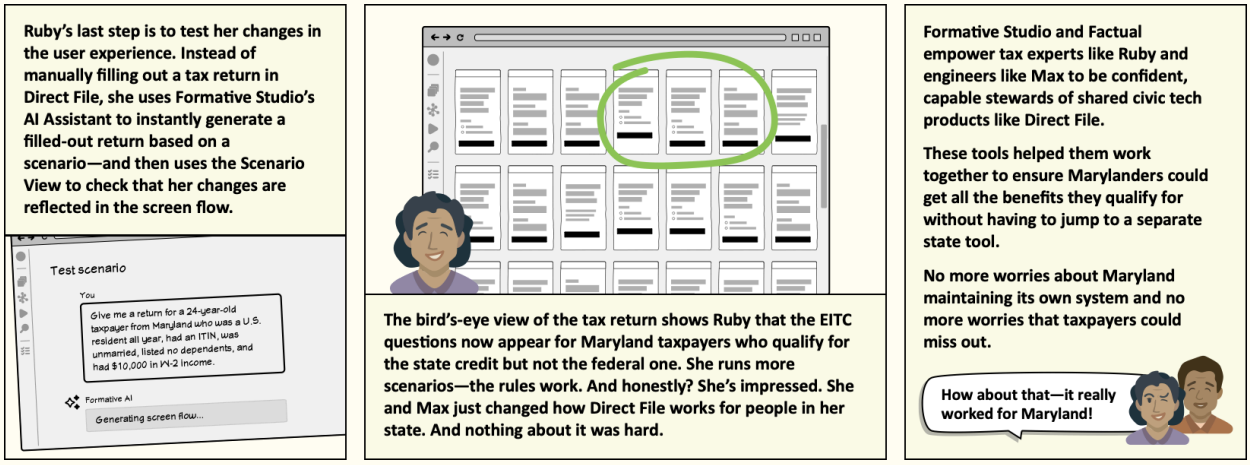
Even better, the changes are simple for Max to implement. He deletes the two lines that need to be dropped, commits the "Qualifies for MD EITC" update, and moves on with his day. Leaving him time for tasks tougher than this one turned out to be.



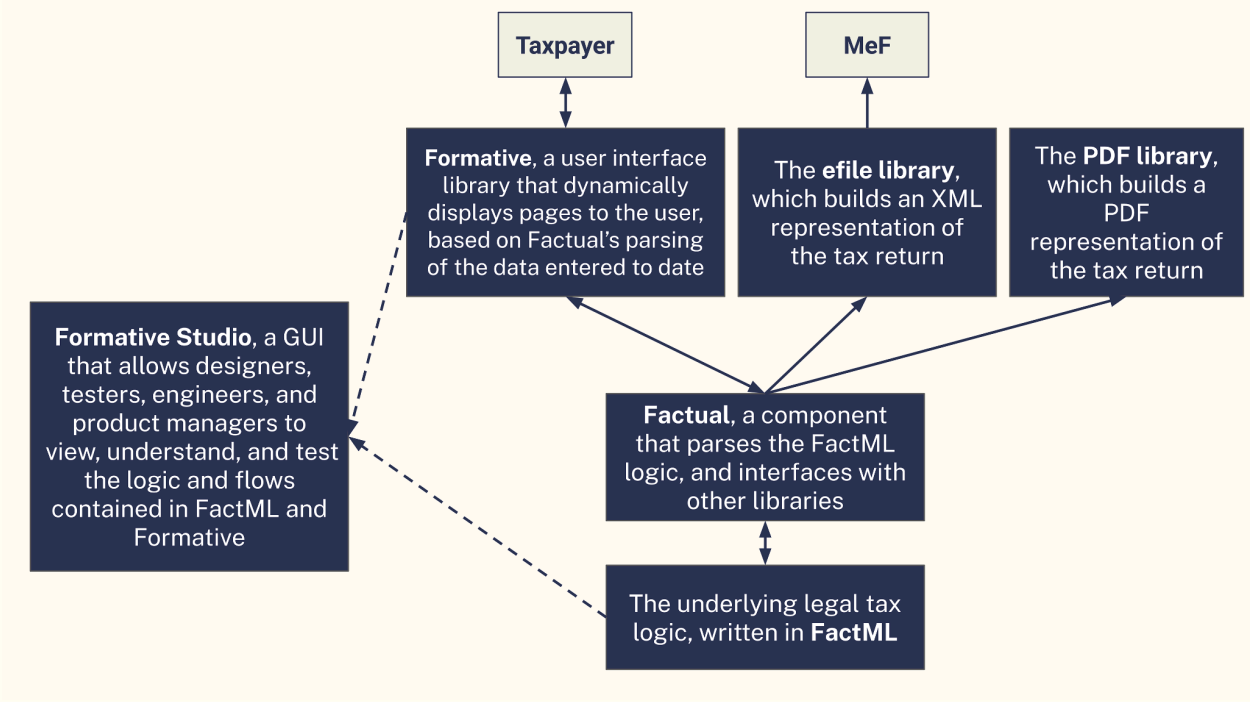
Later on, Ruby checks how the updates turned out.

When she pulls up the new fact, she again sees a plain-language description of what it does.

Max didn't have to write that description—and no one needs to maintain it. AI automatically turned the XML Max wrote into human-readable language for teammates like Ruby.



9.4.2 Components



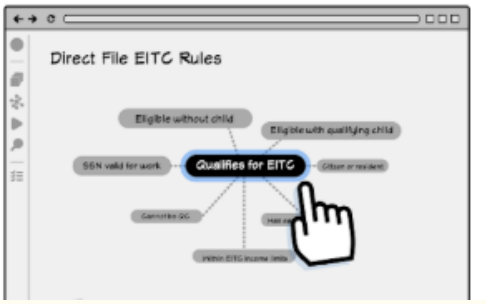
FactML development primarily consists of four components: FactML itself (the fact graph), Factual (a parser for the graph logic), Formative (an interface library), and Formative Studio (the really new element — a back-end GUI that allows the cross-functional team to interact with the functionality).

- **Fact Markup Language (FactML)** is a generalized XML-based specification for defining legal code (that is, statute) as software code (that is, machine-readable instructions), which provides a framework for interoperability between domains and applications.

    - FactML is not the tax code per se; it's a specification for resolving the tax code into a piece of software. In the Direct File context, we would say that the Internal Revenue Code has to be represented *in FactML*. The FactML representation of the IRC contains all the relationships between facts that comprise the tax code.
    - Direct File more or less fully developed a notion of FactML, and has a representation of (relevant parts) of the IRC represented in this XML. The documentation of the specification is incomplete, though, and needs to be refined.
- **Factual**, a portable business rules engine that parses the relationships defined in FactML. Factual validates and derives conclusions from user-provided and imported data, based on the logic written in FactML, and provides information to other libraries in the application.
    - Factual can be said to parse and create an instance of the fact graph. It is the interface between actual data and the logic in FactML.
    - In Direct File, Factual was the component built in Scala, and was relatively robust. (There has been some debate about whether this is the appropriate language for this component.)
    - Essentially, FactML and Factual *together* comprise the much-discussed fact graph logic that underlies Direct File; they are presented separately here since they are, properly speaking, separate components.
- **Formative**, an opinionated interface library for a user-facing application that collects and asserts information, displaying relevant questions and assertions in response to specified goals. In the Direct File case, this is the Direct File taxpayer-facing front end, which has the goal of collecting the information needed to generate a complete tax return.
    - 'Opinionated' means that the library contains a clear set of defaults about what good looks like. For example, Formative has the default that there ought to be one question per page.
    - Formative can be said to take the fact graph, as instantiated by Factual, and create a user interface out of it.
    - In Direct File, this component of the process was less mature. In fundamental ways, the front end was manually specified, rather than being dictated more abstractly by the interactions of the front end designs and the fact graph.
    - By constructing the front end at an appropriate degree of abstraction through Formative, the product team is able to avoid manual kludges and duplicative effort. Team members must specify the tax logic in FactML, and must create the appropriate language and interface in Formative. The complexities of flow and page show logic thereby should be able to be handled nearly automatically.
- **Formative Studio**, a graphical user interface for viewing, annotating, and testing the interfaces and tax logic contained in Formative and FactML. Formative Studio is the linchpin that enables effective collaboration between user experience designers,

subject matter experts, and engineers, by making the entirety of the product tractable in a single interface.

- This component was by far the least developed in the first years of Direct File. In 2025, Direct File introduced the all-screens view feature, which was only the first step toward the fully-fledged product we envision Formative Studio could become.
- Formative Studio fundamentally does *not* let users *edit* FactML, or (at least in its initial incarnation) edit Formative flows. We are not proposing creating a GUI that allows subject matter experts to generate tax logic changes through a no-code interface; a subject matter expert cannot, for example, delete a fact from FactML through the click of a button. Instead, through a series of systematized views of the various components, the subject matter expert can define, describe, and annotate the deletion of that fact, thereby ensuring future team members understand why it was done, and greatly easing the process of communicating to engineers what the requested change is. Engineers, though, would still have to make any literal edits to the fact graph. In the long run, on the other hand, it is plausible that subject matter experts might be able to edit Formative flows (e.g., make content edits to existing pages) in Formative Studio without engineer involvement.
- At its core, Formative Studio allows users to tractably view and interact with the functionality implied by the previous components. Users would be able to view (a) the fact graph (which is to say, the logic in FactML), or (b) the pages of the user interface (which is to say, the functionality of Formative).
    - The graph view is notionally explored in the middle panels of the storyboard in Section 9.4.1, above. It would allow a user to visualize which facts are driving other facts. To maintain legibility, it might show only one or two nested layers of facts at a time, allowing users to click down to additional layers as needed.



        - There might also be the option to view facts in a non-graphical format, e.g., a searchable table containing the name of the fact, and a list of the other facts it stems from and contributes to.
    - The user interface view is a view of the pages a user would see while going through the application, and is notionally explored in the penultimate panel of the storyboard.

            - An early version of this — the all-screens view — existed in Direct File in 2025.
            - Users would also have the option of viewing filtered versions of all-screens — filtered by specific tax provisions (what are all pages that impact CDCTC calculation?), or by specific tax scenarios (for a specified test taxpayer, what are all the pages they would see, in what order?).
    - Accompanying these views would be a scenario library, with saved taxpayer personas that can be created (a) manually in Formative Studio, (b) manually in the Direct File front end, or (c) using AI queries in Formative Studio (e.g., 'create a tax household with these characteristics').
        - At any time, the graph view or the page view could correspond to a particular loaded persona (or to the full set of screens/facts). The data corresponding to the persona could be modified interactively via the graph view or the page view.
    - Annotations could be written and attached to pages or to facts, allowing users to document the intention of different functionality, why decisions were made the way they were, or, e.g., the legal review status of a component.
    - In addition to the core elements, Formative Studio would require functionality to view and test non-flow functionality, like, for example, post-submission messages, or return PDFs.

9.4.3 The impact for Direct File

- By representing clearly and cleanly in Formative Studio the system constructed in the other layers, these tools allow designers, subject matter experts, and engineers to collaborate seamlessly, speaking a common language, and documenting decisions for future team members.
    - This does not mean no-code edits to tax logic; designers can't, for example, go into Formative Studio and create a new fact and corresponding page in the product. But they can use Formative Studio to define, in FactML terms, the use of the page, and to leverage Formative to streamline the process of designing and building the page. And they can communicate these requirements to engineers in a simple, tractable way. In the long run, they might even be able to edit the page content in Formative Studio. This would all radically increase the speed of development.
    - The team members can also use Formative Studio to leave a paper trail of why certain decisions were made, and how further changes should be incorporated in the future.
- Thanks to this common working space and language, states can build their own state tax functionality into Direct File itself, and do not need to maintain separate products.
    - In a world without these robust back-end tools, building pages and logic into Direct File is an essentially manual brute-force process. In this paradigm,

including state tax logic in Direct File would require members of the Direct File team to design and build the functionality; there is simply not enough clarity and not enough abstraction to cordon off a component of the code where states can configure their rules.
    - With these back end tools, it becomes possible for separate state teams to build state-specific components into the Direct File codebase, building on the functionality already in the product.
    - As with the Direct File team, this does *not* mean that state employees can make no-code changes to the product. As on the Direct File team, state subject matter experts and designers would use Formative Studio to define specific changes required in FactML and Formative, which they would document for (state) engineers. The state engineers would build these changes, and submit them as pull requests to the Direct File team. Because of the clarity of the set-up, these tickets for engineers would be far less work than under the status quo, and the pull requests would be relatively easy to review.
        - The Direct File team would be able to define which portions of FactML and Formative flows states can and cannot edit. There would be state portions of the fact graph and state portions of the user interface. But it *would* be one graph and one interface. State provisions, for example, could draw on a variety of facts defined by the Direct File team.
    - In this world, states *would* still need product teams. In fact, they would even need product teams versed in the Direct File development process. But, as explored above, the overall cost to states would be far lower than under the status quo.
    - The integration would create some additional work for the Direct File team, who would have to review state pull requests and interface with state teams who had questions about the set-ups of FactML and Formative, especially in the early years. On the other hand, the state strategy under the status quo required effort from the Direct File team, too, managing API integrations for state tools, monitoring various relationships with independent products, and answering detailed questions about tax provisions that could not be viewed in a centralized back-end. All told, the increase in effort for the Direct File team is probably limited.
- Formative Studio dramatically speeds testing.
    - Testing a product like Direct File, to ensure that the tax return always matches the information a taxpayer enters, is an incredibly central part of the process. It's also incredibly tedious. Subject matter experts need to come up with taxpayer personas that collectively test all of the calculations and distinctions drawn in the software. Team members then need to manually create these returns in the product, and document the inputs that gave rise to them. Testers need to run these personas through the product and carefully review the output for any errors, some of which can be very hard to spot —

especially since it can be hard to reproduce bugs in the user interface. Designing and overseeing a testing regime like this is very challenging at any level of complexity, let alone the complexity Direct File would reach in future years. Direct File — and integrated state tools that used Direct File output as their input — were more or less brute-forcing this entire process.

- Formative Studio would greatly facilitate the testing process. Using the page view by persona, testers can easily confirm that the product is showing the right interface to the right taxpayers. Formative Studio would include an interface to store and annotate personas for further use. It could even use generative AI to create new test personas, rather than having to design and implement them entirely by hand.

9.5 Current status and next steps

Today, in 2026, some successor projects to Direct File, within the IRS and elsewhere, continue to build on these concepts.

It is our view that fully proving out this way of working requires a robust and substantive project in which to iterate and work out the kinks. Product development like this — and especially in the case of Formative Studio, product development is what we are talking about — cannot happen in a vacuum; there need to be actual users and an actual use case.

Direct File was of course the ideal use case. In the absence of its potential reanimation, where else might teams work on these ideas?

1. *Taxes in states.* Building a Direct File-type filing tool in states might seem like the obvious option; and, from a technical perspective, it has all the necessary attributes. But, for reasons we've discussed elsewhere, there is limited need for standalone state tax filing tools in our integrated federal-state tax system. States should not expect many taxpayers to use a standalone state filing product, since most will file their state returns however they file their federal returns. While the use case could prove sufficient for the technical work of FactML development, it's not a product that one would expect to have many users, and as such it might be hard for states to really justify the expense of the overdevelopment inherent, particularly, in the Formative Studio portion of the project.
2. *Taxes in another country.* FactML development could be piloted in the context of an entirely different tax code, in a country that, like the United States, is still lacking a free, public tax filing tool. Canada, for example, is such a country.
3. *Non-tax use cases.* We still hypothesize starting with taxes is advantageous, since designing for the most complex use case (taxes) will most likely create a solution that works for every use case. But starting with another use case isn't prohibitive, and the FactML suite could be developed in the context of a benefits application, or, perhaps better yet, an integrated benefits application for multiple programs.

Teams working in these domains could all serve to advance the ideas we lay out in this chapter. But the right next steps for technologists or policymakers interested in these ideas are admittedly unclear. We are talking fundamentally about a collection of ways of working, rather than any particular policy idea or well-circumscribed product — and technology teams will always be hard-pressed to develop a replicable way of working rather than to simply deliver the project at hand. In practice, bringing the ideas in this chapter to life may end up being a dispersed cultural shift more than a single specific breakthrough.