The Codex Vision - TC Review

NOTE this topic/document was discussed in early 2019 and the Tech Council's conclusion on it was documented here: 2019-03-06 Technical Council Meeting Notes

General conclusion: There are too many open questions and concerns to agree that this document represents the direction we want to head. As written it doesn't. More discussion is needed.

What is Codex?

Codex is a normalization and virtualization layer that allows Folio to integrate metadata about various resources regardless of format, encoding, or storage location. It is the piece that allows disparate resources to be surfaced using a common vocabulary and description.

Normalization. Codex removes differences in encoding and format to provide a single representation of all participating resources, regardless of how they are managed.

Metadata: Codex implements a light-weight (simplified) metadata model to describe resources. This common denominator can be mapped to most existing metadata models, thus providing a common vocabulary.

Virtualization: Codex spans storage locations. It does not matter whether a resource is managed locally or in a remote system. Furthermore, some systems may not be directly responsible for the management of resources, but may be aware of them (e.g. Orders). These systems can also participate in presenting normalized metadata to Codex - essentially providing “pseudo resources” for Folio.

Layer: In a layered representation of Folio, Codex sits at the top. It can be the starting point for all inquiries on resources. From this layer, one may drill down further into the lower, richer layers for any given selected resource.

No really, what is Codex?

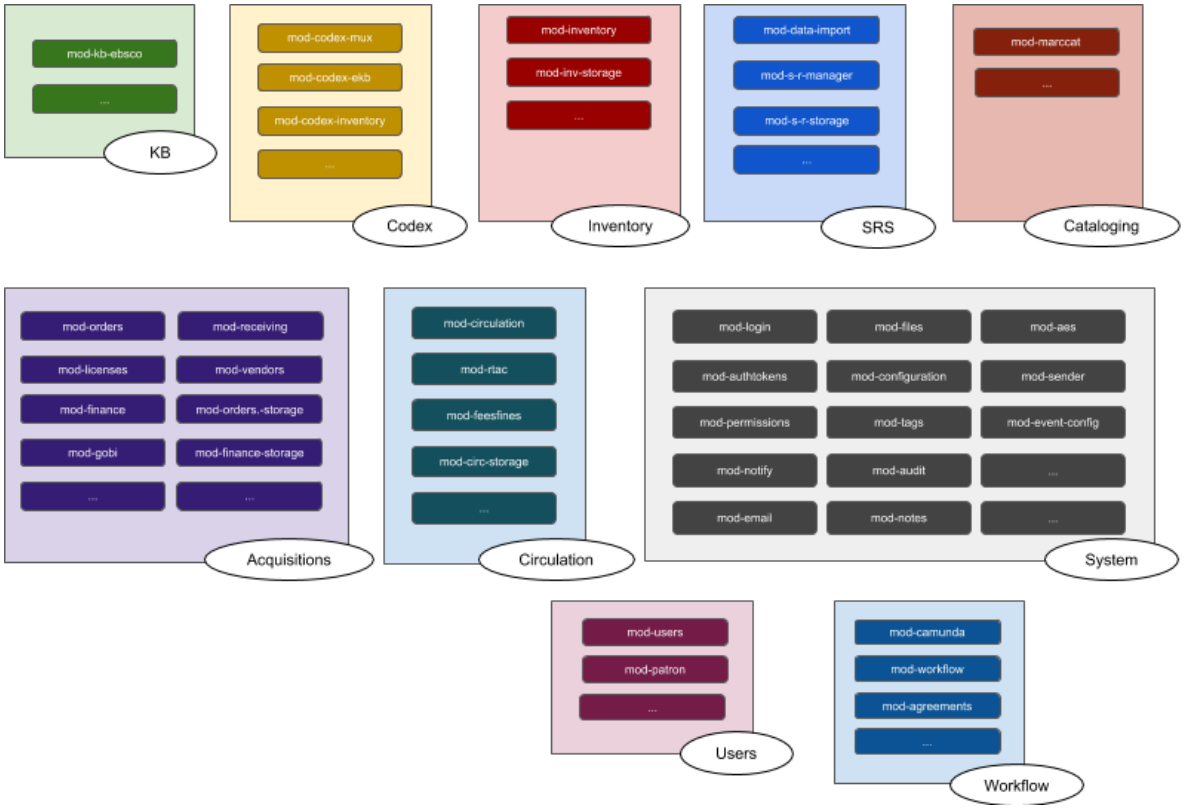
Domains

Folio is a platform that is based on a (specific) microservices architecture. Codex plays a key role in that architecture. A fully featured platform of microservices can easily consist of dozens of individual microservices.

However, some microservices are closely related to others, as shown below using common colors.

Those closely related microservices can be grouped together into what we call a domain. The modules which belong to the same domain will reflect the following characteristics:

- a shared context
- a shared vocabulary
- awareness of each other
- a shared data model



Codex is a Domain

Codex is a domain which consists of multiple modules (and apps). The most familiar Codex app is the Codex Search app. It provides the ability to search for resources across a number of other apps that may be used to manage resources: e.g. eholdings, inventory, institutional repositories, etc… A given app, such as the Codex Search app, typically consists of multiple modules.

But Codex Search is not the only app in this domain. The types of other apps which might be found in the Codex domain include: catalog exports; authority control; resource relationships; agreements apps. Unlike the Codex search app, these other apps may require data, in which case the relevant Codex domain modules would implement their own data storage.

Codex addresses the Entanglement Problem

One of the challenges of a microservices architecture is that of managing the large number of individual microservices that come together to create a coherent solution. For Folio, part of that problem is addressed by Okapi which provides a registration mechanism for modules and the service interfaces they implement. However, it only provides a flat model where all modules sit at the same level.

Folio microservices are implemented as individual modules. In keeping with the principles of microservices, those modules should not need to be aware of the inner workings of other modules - especially across domain boundaries. Each microservice should concern itself with the capabilities that it is responsible for delivering. It should be unconcerned by other similar modules in the system and whether those are present, active, available, etc… Each should just mind its own business.

But in practice, domains, and the microservices they contain, form a system and there are necessary interactions between the microservices, including across domain boundaries. Those interactions create dependencies, and in a flat model, this leads to entanglement - thus breaking the promise of a loosely coupled system.

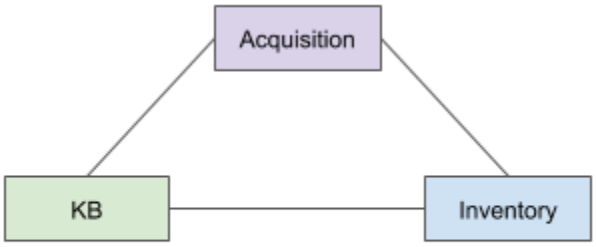
The Entanglement Problem

The interaction between microservices (i.e. modules) is an integration problem. It requires that individual modules be aware of other modules. For modules in the same domain, this is expected and not an issue. However, the same is not true across domain boundaries.

One way to manage interactions between modules is through direct integration. In that approach a module retains explicit knowledge of specific interfaces provided by other modules. It must deal with operational concerns by providing proper error handling and fallback scenarios should those other module(s) not be available, or even present in the system. It must also ensure ongoing compatibility as other modules evolve and interfaces are added, changed or retired. That effort is then repeated for all modules with direct dependencies.

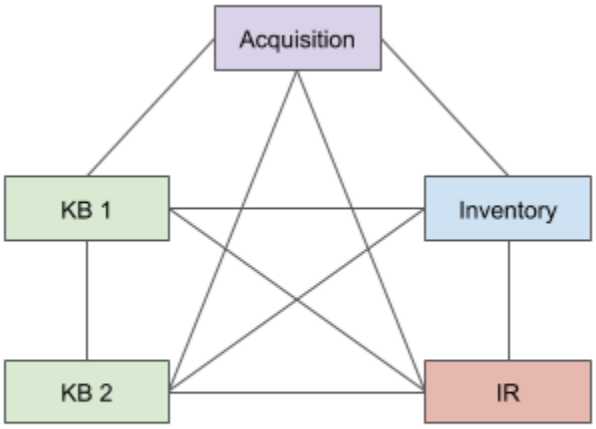
Unfortunately, this direct integration approach leads to a highly coupled system that can quickly become inflexible and fragile. In the particular case of resource management, there are multiple domains involved. All of them would need to interact with the others at some point to deliver various functionalities.

Imagine that a Folio system has 3 resource managing modules (e.g. one from each of the inventory, KB, and acquisitions domains). Each will require 2 integration points, one for each of the other 2 domains. In total there are 3 integration links between the 3 domains. .



3 Resource Managing Modules
3 Integration points

This may seem like a manageable approach that can scale linearly. But it is not. Now imagine that a Folio system has 5 resource managing modules. In all likelihood, in a flat modular system, each module will likely need to be aware of each of the other 4. As shown below, the number of interdependencies grows to 10.

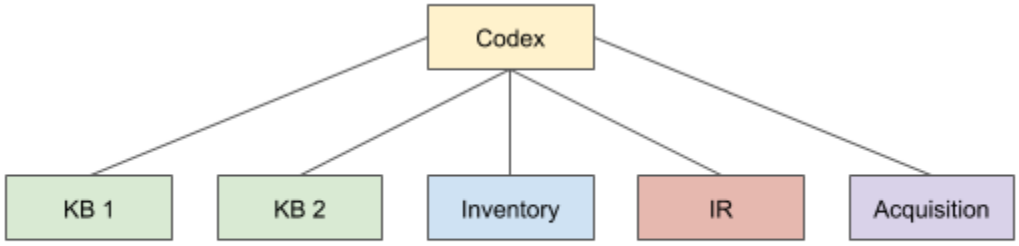


5 Resource Managing Modules
10 Integration points

In the general case, if a Folio system has N resource managing modules, the number of potential integration points isN(N-1)/2, which is not linear, but scales on the order of N2.

A different approach is to use a brokered integration. A new service can act as a coordinator between the different related modules. Each module in turn needs only to integrate with the brokering module. The problem of managing the state and availability of the different services can be achieved in one place. This allows the individual modules to “keep minding their own business” and remain blissfully unaware of other modules’ business. A brokering microservice does need to be aware of the other modules, because that is its business.

Essentially, the brokering module approach introduces a hierarchy. Through the use of a hierarchy, a module only needs to maintain integration with the level above it. It no longer needs to manage dependencies with its peers for the same purposes. With the use of a hierarchical layer such as Codex, the set of dependencies remains linear upon scaling. Individual resource managing modules only need to concern themselves with integration to Codex.

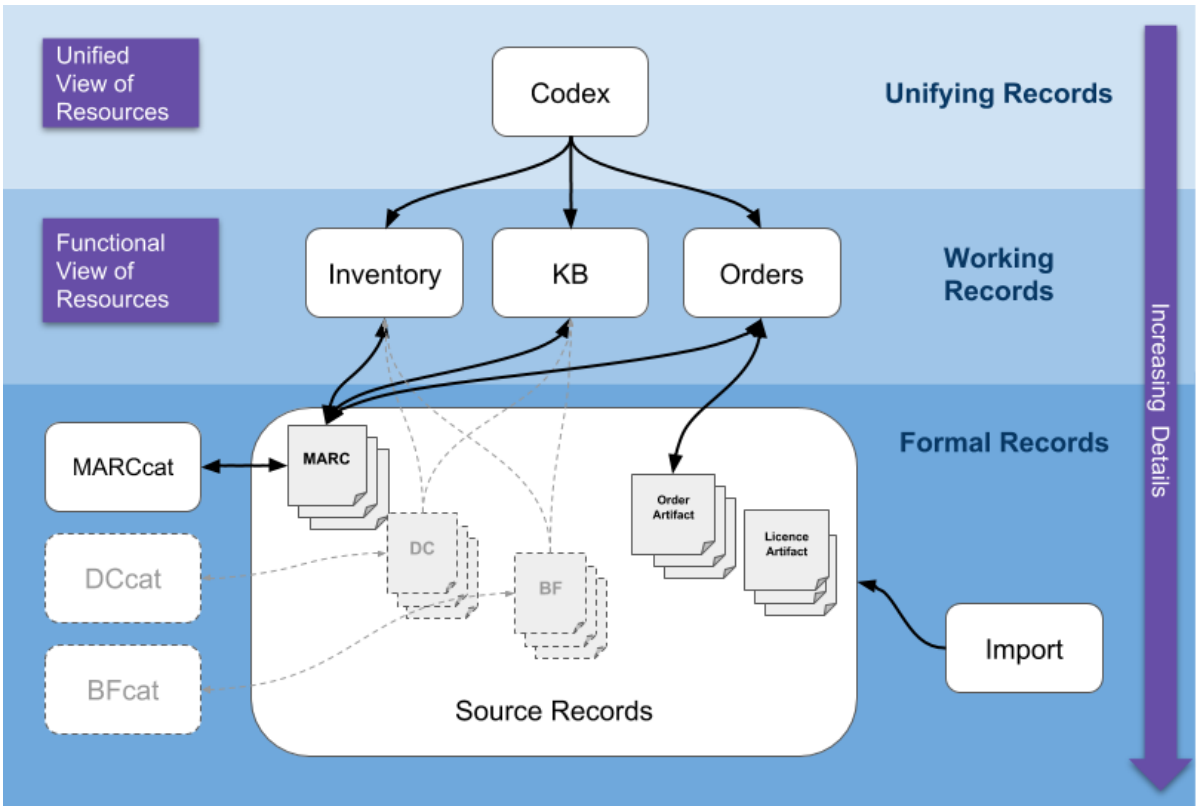


5 Resource Managing Modules
5 Integration points

Codex is a hierarchical domain that acts as a coordinator for resource management. As a coordination domain it is Codex’s business to know about the other domains.

A Hierarchical Domain Model

A significant responsibility of Folio is to manage resources, and more specifically the records that describe those resources. However, not all parts of Folio require the same level of detail for those resources. Nor do they have the same scope requirements: Circulation doesn’t much care about the purchase price of an item, but Acquisitions certainly does. The following diagram focuses on resource management specifically. As shown, the resource managing domains can be organized into layers based on the depth of detail that they require and the purpose they serve.



Starting from the bottom:

Formal Records. These are **the most detailed records available in the Folio system**. This is where MARC records are found in the Folio system. The MARCcat app is used to edit, validate and even create MARC records which live in Source Records. Eventually, Folio will provide similar support for Dublin Core, BibFrame and other non-MARC formats. Note that In addition to bibliographic records, formal records can include such things as Order artifacts (EDI) and License artifacts (e.g. scanned paper licenses). In support of all source records is a sophisticated Import system that manages the ingestion and lifecycle of these documents.

Working Records. These are **the app-specific versions of records that are necessary for the apps to deliver their functionality**. These records may be bound to related source records at the lower layer. For example, an Instance record in Inventory may be bound to a MARC record in Source Records. When bound to a formal record the working record does not need to duplicate all the fields contained in the formal records - it only needs those that are required to complete the functionality delivered by the apps in that domain. Note that working records may also contain additional fields, not found in the formal records, but which are transient or operational in nature.

Unifying Records. This top layer is the one in which Codex and its modules operate. The purpose of this layer is to **provide a uniform and high-level view across all of Folio**. In particular Codex presents a complete and consistent view of all resource related objects from anywhere in the Folio system. Codex is afforded visibility to all the resource managing domains that might exist within a Folio system. It is not to say that Codex contains all the metadata related to any resource within Folio. Instead, resource records surfaced through Codex are linked to working (functional) records in the various resource managing apps, In turn those may also be linked to formal records, thus forming a path from Codex to the finest degree of metadata granularity.

Codex is the Entry Point for Resource Management

Folio is a modular platform. Consequently, functionality may be added to the platform at runtime, as needed, in the form of modules. But it also means that a Folio installation may be created with a reduced number of components: streamlined to provide only necessary functionality.

In the case of managing resources, there are likely a number of different apps whose modules could play a role. There may be an inventory app (or two), and/or an eHolding app (or two) and/or an Institutional Repository (or two). It is always possible to directly go to any one of those apps and access the capabilities it provides. But that assumes a pre-established knowledge that the app in question is actually responsible for managing those specific resources. However, In the more general situation, it is better to start with a Codex domain app as the entry point. From there it is possible to navigate to any part of the system in order to manage resources in the appropriate context.

Codex provides the starting point for locating and managing a resource in any of of the constituent parts of Folio. Once a resource is located, it can be examined in full detail in the appropriate Folio app that is responsible for it (the source of truth). This is the case for the user experience of a person interacting with the Folio UI. But is it also true of any external system required to integrate with Folio for resources! Such a system should not be integrating directly at the individual resource managing module in Folio - since none of them will contain a complete representation of all resources. Nor should the external system need to integrate with multiple modules and then perform its own merge and reconciliation. Instead it would integrate to Codex which is the hierarchical entry point to all the resources located within a particular Folio installation. The external system does not need to know about the internal module configuration within a particular Folio installation. It can delegate that problem to Codex.

Codex is a normalizing Data Model

Since Codex is a domain, it provides its own conceptual data model for describing resources. That model is closely related to the BIBFRAME2 conceptual model and containing metadata fields very similar to those in the Dublin Core data model. The data model supports multiple entities such as Items, Instances, Packages and more, as described here. It strives to be a simplified data model that is the common intersection of the more complex and specialized data models used by the individual resource managing apps. The model is complete enough to suit the task of describing any Folio resource, yet small enough to limit the amount of metadata duplication between it and the other domains.

By implementing its own data model Codex provides a mechanism to normalize resource descriptions across all Folio modules. Rather than creating individual mappings between extant resource data models in each app, the Codex data model is a common intermediary, to which those extant data models can map themselves.

Codex manages Relationships between Resources

As part of coordinating resource metadata between microservices, Codex is also instrumental in establishing relationships between resources, in particular when those resources might be described (source of truth) in different microservices.

This establishing of relationships between metadata managed in different services is simply an application of the concepts of linked data. Establishing a connection between two such resource descriptions allows them to remain fully within their respective managing microservices. It provides a solution to the dreaded practice of duplicating and copying metadata between system components.

When a relationship is established between two or more resources, that relationship needs to be persisted somewhere. The sensible place to store the relationship and its links to resources, is in the Codex domain, in the app used to manage relationships.

For example, to describe a particular Work it may be necessary to establish a relationship between an ebook managed in eHoldings and a print book managed in Inventory. The work is itself an entity and therefore it needs to be persisted in a microservice that will be responsible for it (source of truth). One approach might be to attempt to duplicate the ebook metadata in Inventory or alternatively duplicate the print book metadata in eHoldings. However, either of those creates direct dependencies and entanglements between the two microservices. Furthermore, it would introduce a problem of data synchronization and might also create an ambiguous source of truth. A better approach is to use another microservice, which is neither Inventory nor eHoldings, to persist and manage the Work. That microservice would be Codex.

Codex is Resource Central.

Authorities and controlled vocabulary lists are a natural fit for Codex. Implementing these in the Codex domain makes them available to all Domains equally - they can be linked to from Inventory, eHoldings or MARCcat. These could not only be usable in the narrow context of resource management, but also potentially in other areas of Folio..

Codex includes Codex Search

The most familiar use of Codex is to provide a unified search function for resources: Codex Search. Codex supports a generic resource metadata model based-on Dublin Core. In order to participate, resource managing components (i.e. microservices) will implement Codex Search conforming APIs. The integration requires a metadata mapping - a data crosswalk between their native metadata resource representation and the Codex metadata model's representation. By implementing, then registering these API interfaces with Okapi, the resource managing components become contributors to the Codex Search functionality.

The Codex Search functionality can thus provide a single entry point to locate any resource throughout the entire Folio system, regardless of the specific module responsible for managing it. The use of a simplified but generic metadata model is in keeping with the Codex goal of understanding resources regardless of their local format or encoding.

It is expected that specific resource managing modules may provide their own more advanced search capabilities. This is possible since at the level of those modules there exists richer and more specialized metadata not available from Codex Search metadata model. One might start with Codex Search to locate a desired resource and its managing microservice, then drill down from there to more advanced searching.

Codex Searching can also be used as an embedded component to allow other Folio apps to provide a user experience for locating resources, in any part of Folio, to be linked to from another microservice or module. For example, an Orders app may include the ability to conduct an embedded Codex Search to locate and link a resource in a Knowledgebase for selection and inclusion in an Order.

A fully implemented Codex Search will allow searching for all forms of resources: instances; packages; providers. It will also return the holdings and status of returned results within the context of the Folio installation.

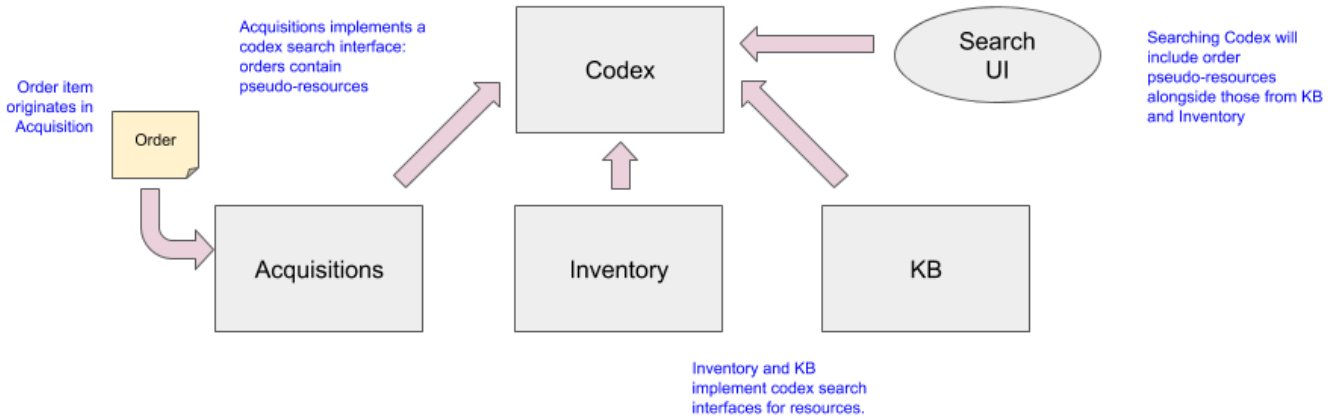
Use Cases

The following selective use cases are illustrative of the Codex concepts discussed above.

Use Case 1: Locating Resources

Folio is a highly modular system, where separate apps might be responsible for managing different resources. But modular also means that specific resource managing apps may or may not be available in a particular Folio system. In keeping with the microservices approach, the individual resource managing apps should also mind their own business and be as little aware of each other as possible. This is where Codex comes in: the business of microservices in the Codex domain is to know which resource managing apps are present and how to interact with any of them. (Codex operates at the unifying layer.)

How to tell if a particular resource is available to an institution, either available for circulation or for access or even just on order?



- In this example there are 3 domains shown, each of which might represent any number of resource managing apps within.
- None of the individual resource managing apps has a complete picture of all resources available in the system.
- Codex is the starting point for locating a resource. Each of the resource managing apps “registers” itself to Codex so that resources may be located.
- The resource managing apps inform Codex of the matching resource for which it is responsible..
- Once the resource is located, the user can drill down into the responsible app to retrieve full details about that resource.
- **The Order system is a resource managing app within the acquisitions domain**.

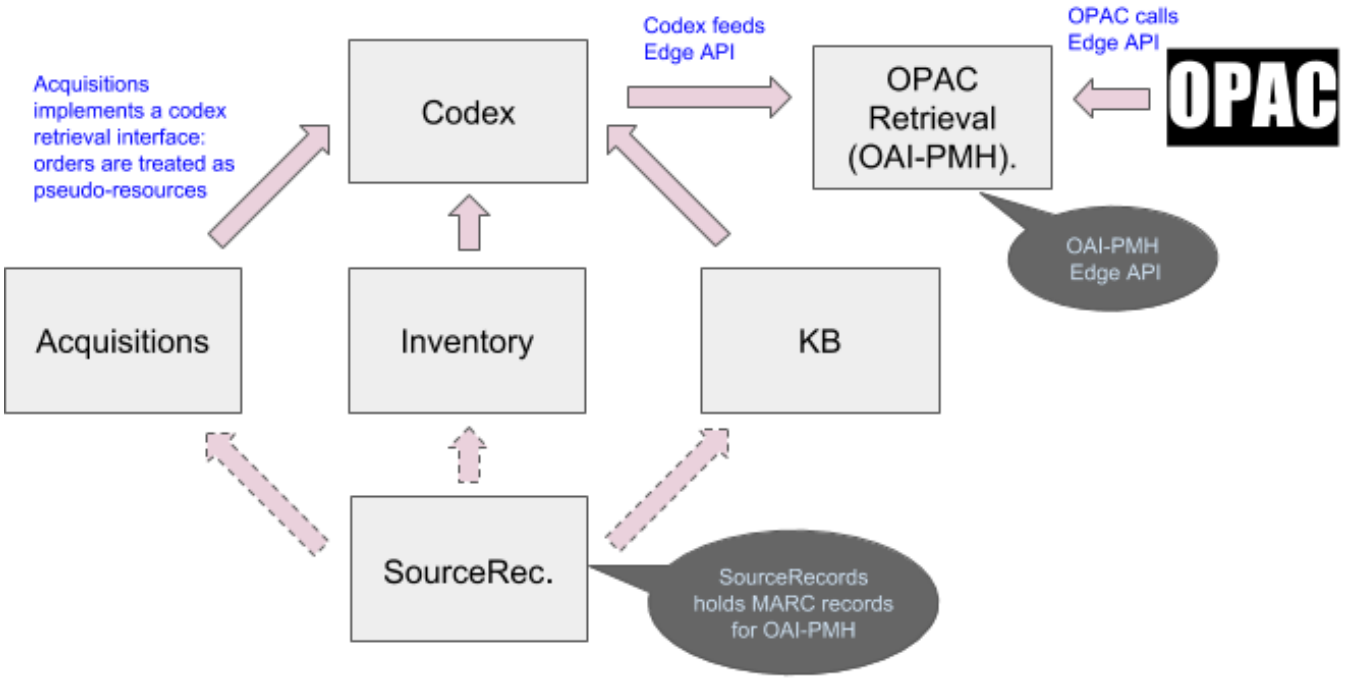
That last point deserves further explanation: the Order system is a resource managing app. This states that there are resources that naturally exist as part of the acquisitions domain. When an order is constructed it will describe through purchase order line items, the “things”,(i.e. resources) that are being ordered. These are resources in their own right and they exist in the Acquisitions domain. Furthermore, it is the Order system’s responsibility to manage those resources until they are received. Therefore, the Orders app can report those resources to Codex when asked to do so. After a “thing” has been received, it is no longer the responsibility of the Order system. A new resource will be described by Inventory or the KB or whichever part of the system is now responsible for managing it and reporting it to Codex. The Order system can now simply stop reporting that resource to Codex because it is no longer its responsibility.

This is a much simpler and more elegant solution than more traditional approaches whereby pending orders might create temporary records in Inventory. In this case, the Order app does not need any awareness of the Inventory system. It does not need to know how to create a stub record there. It does not introduce direct dependencies between those two systems.

Use Case 2: Exporting comprehensive Folio catalogs to an OPAC or Discovery system

It turns out that the problem of exporting a comprehensive catalog of all resource holdings to an OPAC is very similar to that of a Codex Search. The primary difference is that the actor performing the task is another system rather than an end user. An Edge API takes the place of the UI in making calls to Codex.

How do I synchronize my catalog with my OPAC or discovery system?

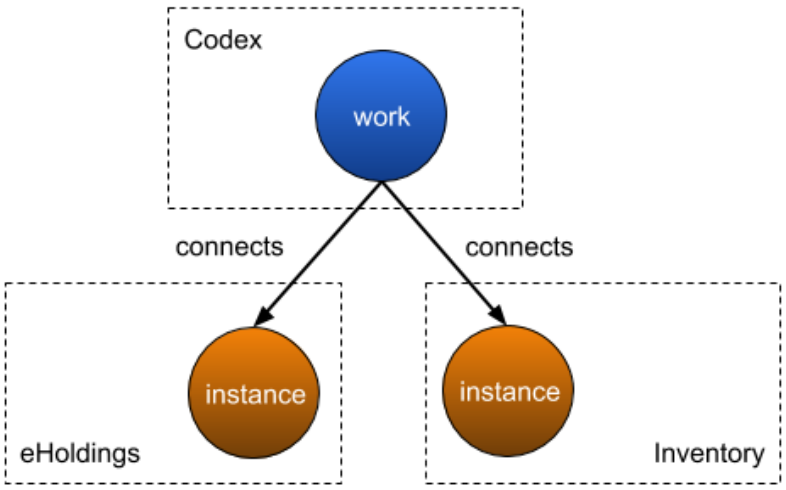


- In this example there are 3 apps shown, but there could be more, such as additional KB apps, or Institutional Repository app(s) or even another Inventory app.
- The SourceRecord Manager holds the MARC (or DC) records describing the resources in the Catalog.
- None of the individual resource managing apps have a complete picture of all resources available in the system.
- Codex is the starting point for gathering resources to be exported to the OPAC. Each of the resource managing apps “registers” itself to Codex so that its managed resources may be aggregated into the OAI-PMH response.
- OAI-PMH requires resource descriptions in MARC (or Dublin Core) format. These will be found in SourceRecord storage. The resource managing apps will contain links to the appropriate bibliographic records in Source Record Storage (SRS).
- An Edge API is used to deliver OAI-PMH responses to the external system. It allows the remote system to integrate to Folio using existing conventions (OAI-PHM).
- The OPAC can be made aware of pending orders since the Order system can surface resources (pseudo-resources) that are “on order” - just as in Use Case 1 above.
- There are no direct dependencies between the various functional modules (inventory, eHoldings, Orders) since all connections go through Codex.

Use Case 3: Creating Associations between Resources

Codex is not just a search interface. There exists the need to create relationships between resources. Perhaps an ebook managed through eHoldings needs to be related to a print book managed in Inventory because both represent the same work. In keeping with the concepts of Linked Data we would want to create a Work object and link it to the different instance records that represent manifestations of that work.

Where do I store the relationship between two different records?



It would be simple enough to create a structure to represent and store such a Work in Inventory or likewise in eHoldings. But the problem arises when the two instances to be related, exist in two different parts of Folio: in this example, in Inventory and in eHoldings. Do we create the work structure in Inventory? Or do we do it in eHoldings? Or do we do it both? The answer is neither. We create the Work object in the Codex domain and link it to the resources that are managed respectively in Inventory and in eHoldings. This avoids a direct entanglement between Inventory and eHoldings. In this case, both can remain blissfully unaware of each other. Furthermore, since we are linking and not duplicating resource records between domains, we also avoid the problem of data synchronization.

Codex can connect resources from disparate parts of Folio and create arbitrary relationships between them (not just Works). For example, it can be used to create and manage categorizations. Similarly, the Codex domain can be used to manage and make available Authorities.

An important consequence here, is that In order to support relationships and authorities, the Codex domain requires its own storage capabilities for persistence.

Use Case 4: Inter-Folio Functionality

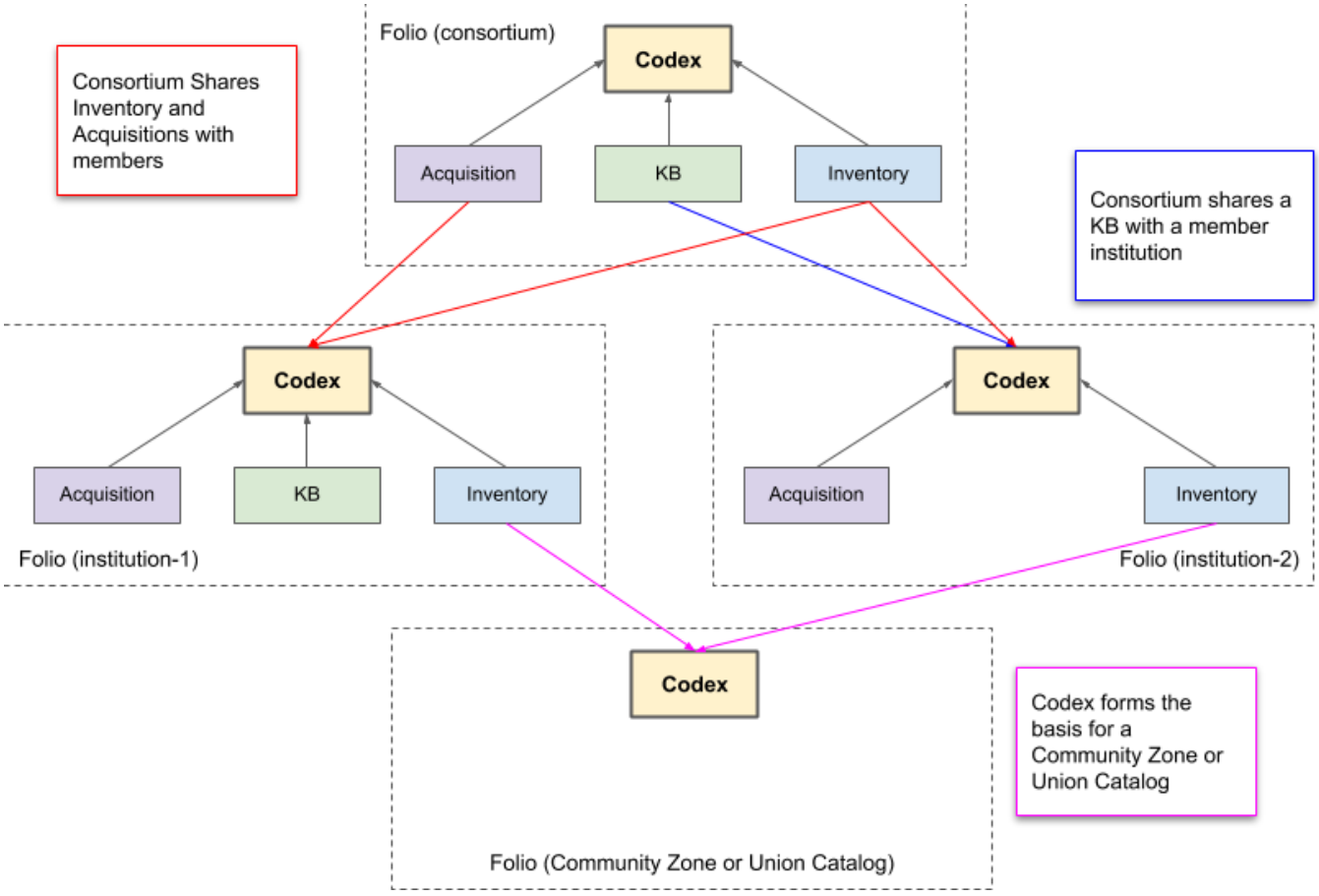
In the not too distant future, there will exist multiple Folio installations. For a number of reasons these will not want to live in isolation, there will be a need for collaboration between them. Motivations include: participating in a consortium; sharing holdings as part of a union catalog; direct interaction with another institution; a shared cataloging effort. From these a number of related use case scenarios can be identified.

- An institution wants to make its resources available to other libraries
- A Consortium wants to make its resources available to its members
- An institution wants to participate in a “community zone”
- A Union Catalog needs to be created

Each of these, and more, may be treated as separate use cases. However, there is a commonality between them all, which is that they concerns themselves with Folio as a whole.

It is only a small leap from a Codex domain pulling together resources within a single Folio installation, to a Codex domain pulling together resources that are located in distinct Folio installations.

Each Folio installation has its own Codex domain consisting of relevant Codex apps. From this perspective, Codex is not only the starting point from which to start resource management within each Folio. It is also the point where each Folio can pull in Codex compatible components from other Folio installations.



Multiple scenarios are Illustrated above:

- In red, a Consortium shares resources with its members
  - The consortium makes its Inventory available to the Codex in each of the member institutions. Institution-1's and Institution-2''s Codexes (Codices) each pull-in the Inventory from the Consortium Folio. They each effectively have two Inventories to work with: one local and one shared.
  - It makes its Acquisitions selectively available to the member institutions. In this case Institution-1 has access to both a local Acquisition or the Consortium's Acquisition, through its own Codex.
  - Integration can be selective. As shown here, Consortium Acquisitions may not have been made available to Institution-2 or that institution may have chosen not to integrate it to its Codex.
  - Similar forms of sharing can be envisioned for other Codex compatible contributors such as Institutional Repositories.
- In blue, a Consortium subscribes to a commercial KB, which it makes available to a member institution which does not subscribe on its own.
  - Institution-2 has integrated the Consortium's KB to its Codex.
  - Institution-1 already has its own KB subscription on its Codex, so it does not need to integrate the Consortium's KB.
- In purple, a union catalog is created that pulls together the inventories from Institution-1 and Institution-2.
  - The union catalog itself could be instantiated dynamically through a dedicated, minimal Folio installation: no Acquisitions; no Circulation; etc…
  - The union catalog is powered by what would likely be a dedicated app in the Codex domain.
  - The union catalog Folio installation would use its Codex to pull-in resources from the remote Folio installations.