Model Driven Architecture

Extend Your Core – Extend Your Future

FUTURE PROOF

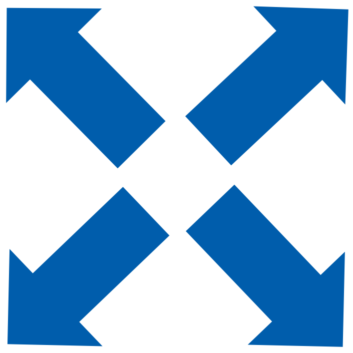
Extensibility is the ultimate in future proofing. Modern design allows banks to rapidly and easily modify the Finxact Core as a Service with proprietary functionality. Using Finxact's JSON Schemas, banks can modify and add components, data elements, generate new APIs, and create or enhance most system objects.

RAPID API CREATION

Banks can quickly modify and add new APIs simply be adding or enhancing Finxact's reference JSON Schemas.

CODE GENERATION

Each enhancement is recompiled and available to Finxact's Core as a Service at runtime, using Finxact's novel versioning methodology.



- Finxact's novel engineering accommodates extensibility from multiple sources
- Custom authoring features for a bank's proprietary functions and extensions
- Build upon Finxact's API library and reuse Finxact's components and/or your own creations
- Modify, enhance and create source code, API, and database elements with versioning and run time code generation

Using modern design and novel engineering Finxact is advancing the state of the art in development. But until we can predict the future, even the most advanced engineering may not anticipate all future configurability needs. That is why Finxact created Model Driven Development. Finxact's novel Model Driven Development provides banks a way to rapidly enhance their core, adding functionality, data elements, and new APIs – thus, future proofing their core system, making configurability always current and making obsolescence… obsolete.

Finxact's Extensible Schema allows for the Finxact core application schema ('base schema'), as delivered, to be extended and/or combined with other schema files provided by any combination of Finxact, customers or 3rd party providers. The combined schema definitions are then used to generate a virtual 'Integrated Schema' definition which is consumed by the code generator to product the supported output definitions.

Finxact
CORE AS A SERVICE

Model Driven Development

The 'base schema' can be extended or customized by simply adding additional schema classes as well as adding or modifying properties on existing classes and providing them in separate JSON schema files. These files are stored in directories that are linked to the base schema via configuration specific search list parameter.

Any number of directories can be included in the search list and the list is processed sequentially. Simply put, given a directory search list "[A,B,C]", the schema definition builds up, beginning with the content of directory "A", then searches "B" and "C" sequentially. If a *.json filename does not currently exist, it is simply added to the schema. If it does exist, its contents are merged with the existing file (i.e.., new property definitions are added to the class and pre-existing property definitions of the same name are overlaid). Currently there is no schema directive to remove or delete a property, but that could be added.

The search list is defined in an OS environmental variable defined in a configuration file ($ZSL?). It is a JSON formatted array and can include complete directories and or files within directories. For example the search list [A, B: [B1, B2], C] would start with directory A, then overlay the files B.B1 and B:B2 then all of directory C.

The separation of the base schema from the extensions allows for easy code management and reconciliation, as the base schema is untouched by any changes (and easily reconcilable). It is also quite easy to add directories and or files to the search path as required, and easy to identify the schema extensions by listing the directories they reside in.

This extension model could be considered 'unsafe' in that it is possible to introduce changes to existing schema definitions that generate errors or break existing core application code. It is also possible to introduce new schema elements that also break the application or affect its behavior.

On the other hand, the partitioned and isolated nature of the content provides for very simple content management and an easy process flow to both introduce changes and to revert to prior versions (via modifications to the search list). It also treats all schema content as a 'first class citizen' in the application as the resulting generated components are produced from a combined definition consisting of all of the component definitions. It is also quite easy to design a directory structure that can be implemented for every application instance, offering easy and isolated extensibility at the instance level. It is even possible to share schema content (e.g., the base schema and Finxact patches) across instances by including common directory references in the search list.

Finxact
CORE AS A SERVICE