Improve Direct-Feed API Integration Projects to Generate Strategic Benefits

Intuit Implementation Support Team

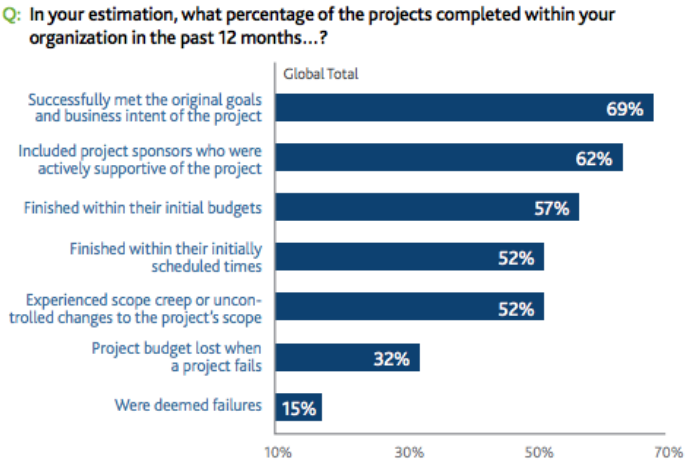
Document Date: 2020-06-03

Table of Contents

Executive Summary

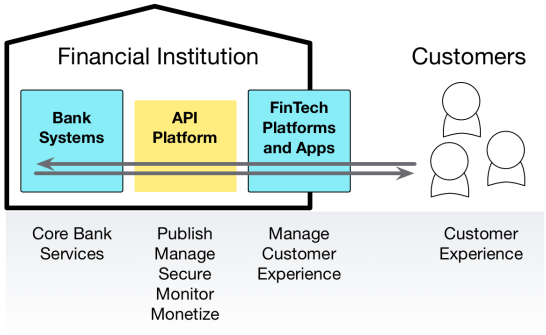
Improved partner integration provides strategic benefits and reduces project time and uncertainty. Integrations that connect partners offer special challenges, because they compound the usual issues that arise in software development. Smaller issues can often be worked around, while bigger issues can impair cost or timeliness, or even result in a failed project.

Project Management Institute (PMI) surveys indicate the scope of the problem.* Despite significant changes in project management over the past 20 years, 52% of projects still overrun their dates. 52% of projects experience scope creep.



An API (application programming interface) is a set of tools and protocols that specifies how parts of a solution interact. APIs are a key element in today's evolving digital toolbox. APIs can provide significant strategic advantages in partner integrations:

- They offer more benefits from services and solutions.
- They maximize solution performance and improve competitive factors.
- They improve implementation velocity and help reduce uncertainty.



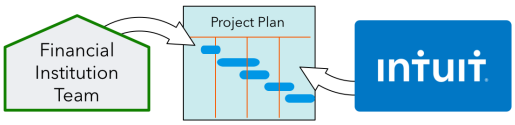
Some highly successful businesses market API-based solutions built off of other services or solutions. For example, Marketo (MKTO) is a popular add-on for Salesforce and Microsoft CRM. It sold for $1.79 billion in 2016. Their successful API integration into two of the largest CRM solutions adds value for both solution providers and solution customers. The CRM providers win by selling a desirable expanded feature set to their customers, without having to develop that functionality themselves. Customers benefit from a highly focused and reliable solution that more closely matches their requirements.

Build Effective API Integration Projects

API integration projects are time sensitive and mission critical, so what can you do to help these projects meet stakeholder expectations and ensure customer delight?

Align the Teams

Develop a Joint API Project Plan



If both teams are aligned on the integration requirements, they can ensure they're working toward shared goals. Even in environments where each contributor owns its own piece of the integration, you can build a joint project plan, highlight the risks, and track it effectively. The most important reason to use a joint project plan is to draw attention to issues that change delivery timing. Plans always need to evolve over time. Recognizing that a plan is likely to need to change at least 49% of the time will help to keep the project aligned and focused.

Four Keys to Integration Projects

Intuit finds four key items deliver the most benefits in API integration projects. They remove blockers from the process, improve quality, cut down on meetings, and ensure timely delivery:

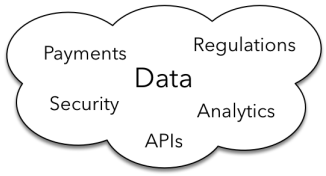
- Adopt a Standard (for Authorization of data access and a Data retrieval API)
- API Documentation (developer portal, swagger files, sample responses etc.)
- Pre-production Test Environments
- Communication and Problem Resolution: The Intuit Readiness Assessment

This white paper discusses how each of these can improve your integration projects.

Adopt a Standard

An API is not a new thing, but API value has increased as technology has grown more complicated. In the same way, customers with always-connected devices expect that they will be able to interact with your services from that device, at any time, with strong security and exceptional functionality. API-based FinTech solutions offer you a way to evolve and expand to support those customers. Implementing an API to provide secure access to information held in internal systems is a critical undertaking, and you want to get it right. Reinventing the wheel is an expensive and potentially fraught problem.

When you adopt a standard, you improve the speed of adoption of your API, because partners will interpret the standard as a way to reduce risk. In the best case, your partners have already built solutions using that standard, which can help reduce complexity and avoid problems. Standards typically have basic pitfalls teased out already, and if issues are identified later, you have resources within the standards body and the usage community that you can use to resolve them quickly.



Partners can also be a source of information for best practices. They can provide free advice about the best ways to approach something. This is valuable because standards are prescriptive about structure, but don't always define implementation details.

For financial data exchange, Intuit recommends using OAuth 2.0 for Authorization, and Durable Data API for exchanging data. These APIs are the collaborative efforts of many to provide a well-lit path for others to follow.

- OAuth 2.0 - https://oauth.net/2/
- DDA - https://financialdataexchange.org/pages/dda

Alternative Authorization standard:

- OpenID Connect - https://openid.net/specs/openid-connect-basic-1_0.html

Other standards that may be used for data exchange are:

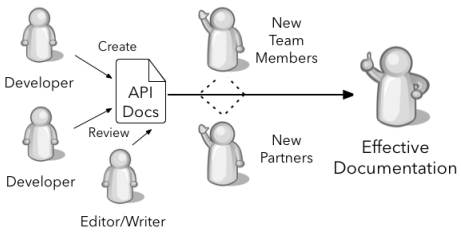
- Open Banking - https://www.openbanking.org.uk/read-write-apis/
- OFX 2.2 - http://www.ofx.net/downloads/OFX%202.2.pdf

API Documentation

Although we may dislike the effort that goes into documentation, a documented API is the easiest and fastest integration path. Without good documentation, partners spend too much time trying to understand how the API works, and don't necessarily grasp the full benefit. Typically, API partners think APIs are self-describing or discoverable, but defining the wider context removes a lot of risky and expensive trial and error.

Your API implementation time will improve dramatically if you document how it works and treat any partner like a newly hired employee, exposed to the API for the first time. Partners should be able to view documentation that highlights the process required to effectively use each feature of your API. Being clear and complete will save time during the implementation.

API Documentation Development



Easily Available

You can improve documentation effectiveness by making it widely available. Try to think of ways that you can improve access to your documentation. Hosting on a developer portal provides the most direct benefit, because you can align your documentation with other API integration project elements in a one-stop shop.

If a developer portal is not easily available, a comprehensive document covering all the above, combined with a well-structured shared documentation platform such as Swagger, will give developers a lot to work with. For a good example of Swagger, go to http://editor.swagger.io/ - / and load the default Pet Store project. Keeping your documentation current makes it easier to keep the implementation on track.

Essential Elements

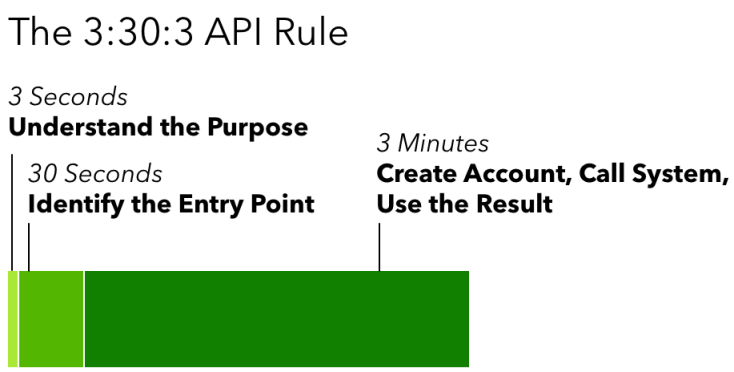
For each endpoint, you should at a minimum document:

1. Its purpose
2. All parameters
3. Response content type
4. Response code
5. Response body
6. An example of the call
7. Error flow

For extra credit, include example calls in different languages (cURL, Ruby, Node.js, Java, PHP, Python). This will improve your documentation quality and help avoid costly delays for questions.

Having all of these elements in place will help you meet the 3:30:3 rule*, according to which developers need to be able to:

- Understand the purpose of the API in 3 seconds,
- Identify its entry point in 30 seconds, and
- Create an account, call the system, and use the result in under 3 minutes.



* Pekelman, Ori (2012). Lipstick on a pig. How (not) to design a modern API over legacy systems. Presented at APIdays 2012. http://pekelman.com/presentations/apidays/

Special Requirements

After you define the essentials, you must document any special requirements of the API:

- Specific implementation parameters, whether required or optional.
- Supported scopes.
- Special requirements for calling your API, such as
    - Mutual SSL
    - Whitelisting
    - Cookies for Load balancing
- Returned error codes and messages from your API.
- Specific scale and performance SLAs for your API.
- Your versioning plan and roadmap for future development.

API Documentation Examples

Remember that your objective is to make it as easy as possible for your partners to work with your API in an integrated solution. Clearly describe flows through the API to expose specific features and speed up the implementation. In most cases, you will start with authorization, and explain with examples how your partner would connect to your API. If it is an OAuth connection, it might look something like this.

1. Obtain your Authorization Code using the following URL

```
https://www.example.com/api/1.0/authorize?
   response_type=code&
   client_id=CLIENT_ID&
   redirect_uri=REDIRECT_URI&
   scope=accounts_and_transactions&
   state=1234zyx
```

2. With the user approving the access, redirect with the auth code and state included

```
https://REDIRECT_URI/?
   code=AUTH_CODE_HERE&
   state=1234zyx
```

3. Swap the Authorization Code for Access and Refresh Tokens

```
https://www.example.com/api/1.0/token?
   grant_type=authorization_code&
   code=AUTH_CODE_HERE&
   redirect_uri=REDIRECT_URI&
   client_id=CLIENT_ID&
   client_secret=CLIENT_SECRET
```

4. Server will respond with tokens and expiration times

```
{

   "access_token":"ACCESS_TOKEN",

   "token_type":"bearer",

   "expires_in":600000,

   "refresh_token":"REFRESH_TOKEN",

   "scope":"read

}
```

OR will return an error

```
{

   "error":"invalid_request"

}
```

After you've defined the authorization process, working with the API should also be clearly documented.

You can see some good API documentation examples online:

- https://developer.intuit.com/docs/0100_quickbooks_online/0100_essentials/000500_authentication_and_authorization/oauth_management_api
- https://help.shopify.com/api/getting-started/authentication/oauth
- http://help.getharvest.com/api-v1/authentication/authentication/server-side-apps/

Test Environments

After you complete your API documentation, you can plan the integration and build solution code (software releases) to your implementation specifications. A test (e.g. "Development", "System Integration Test", "QA", etc.) is a safe environment with representative sample data that uses the same code (software API version) that will be deployed in your production environment. Implementation teams use the test environments to become familiar with the API without risking actual customer data or production systems.

Why test an integration?

Your test environment is a live-action advertisement for the API. Architects and consultants often work in test environments to see how the integration will work for their products and services. Test environments help identify feature gaps or undiscovered requirements that they need to address. If they choose to, they can compile comprehensive design documents or stories for the implementation.

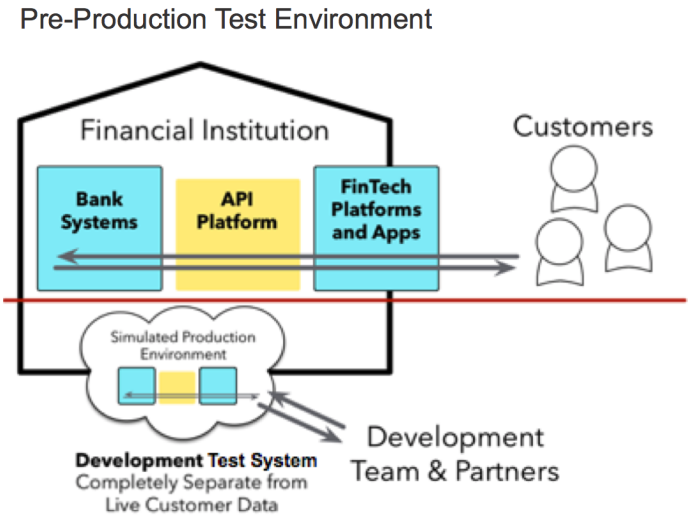
The development teams can take these documents or stories, build estimates, and lay out detailed project plans. A program manager or project manager can feed these details into a comprehensive plan to help coordinate stakeholders. After the project begins, these details help keep things on track.

Even better, because the sandbox works like a production environment, as the application is built, development teams can build functionality and run test cases against the sandbox to sign off. They can exchange real-time feedback with stakeholders, and changes can be tested and approved quickly. If the team encounters issues with the API, they can engage external parties with low risk.

Representative of Production

For the test environment to truly be effective, it should use code that is either planned to be released to production or matches the current production environment, in a well contained environment that provides a high fidelity of the full system. The test environment should have full functionality (integrators can add, update and delete data or can see their activity reflected).

http://editor.swagger.io/#/Petstore is a fully functional test environment that allows you to call the API and add, update or delete data. This is public, so you can see all the sample data people have submitted.



Because the test environment is a controlled environment with representative sample data, you could build it with a simpler authorization method, such as an access token with a long expiry (e.g. 24 hours). Ideally each user of the test environment will be contained into their own area provisioned data (online credentials, accounts, and transactions). Implementers should be made aware of the visibility of any test data supplied.

If the production API is one that delivers data on a regular basis (for example, Bank Account Statement Data), you should set up your test system to cycle a week's worth of data. This allows the implementer to validate processes around processing data sequentially or detecting duplicates. Remember that your pre-production test environment is most effective for API integration development if it offers functionality that is as close as possible to the full production system in use.

Communication and Problem Resolution: The Intuit Technical Assessment

Technical Assessment Phase

Intuit conducts a Technical Assessment (TA) that determines the Partner FI's readiness to fully integrate with Intuit, covering Setup, Data, and API code. It is used to identify and remove major surprises from the implementation process (exceptions to the Standards, specific Partner FI requirements, limitations, etc.), and to prepare the construction of a joint Implementation Plan so stakeholders all have a common view of the project. This helps provide tracking so stakeholders know if they need to make decisions due to delays or other impacts.

The **TA process scope** includes:

1. End User Interaction for Authorization (User Interface, text, consent, logos, etc.)
2. Data
    a. Contract Data List
    b. Documentation Swagger (JSON) file (API spec detailing responses)
    c. Test credentials (online profiles)
    d. Representative set of financial accounts and transactions assigned to test credentials
3. API testing
    e. Authentication (Token Exchange and Refresh)
    f. Data API (Call pattern to use to retrieve an end user's data)

Implementation Phase

The **Statement of Work** lays out the plan for the following phases:

1. Delivery of any required implementation functionality by either Intuit or the Partner identified during the Technical Assessment.
2. Integration Configuration
3. Integration Testing (Component Validation Testing, use of Intuit products in pre-production to see the partner's data reflected, aka 'end-to-end' testing)
4. Integration Pilot (Each Intuit product, incorporating requirements from the Partner)

5. Migration waves (Each Intuit product, incorporating requirements from the Partner) successive set of current users migrated (increasing in both frequency and number of users per wave)

    5.1. Initial Wave

    5.2. X number of follow-on waves

    5.3. Final Wave

The **Readiness Assessment** validates that all the required components exist, and it begins with the following pre-requisites:

- Completed Data List
- API Documentation
- "Sandbox" API environment

The **Data List** is especially important for two reasons. It ensures that the required data and its format is identified and agreed to early in the process, and verification work can proceed using this documented scope as the baseline.

Communications

Every communication should emphasize clarity and should reflect both the success drivers and any expectations that have been set. For example, if the API provider is still in the design and build phase, but sets a rigorous inflexible schedule, issues will arise if there are delays in getting to deployment. If the status of the API is not shared within the business or with the partner, this creates a lot of miscommunication between stakeholders. Missed communications create cost and can put the project in jeopardy.

Communicating up front around the status of the integration project, and where dependencies lie, within or between the partners creates a solid place to start. It is fundamental to the success of implementation projects that the status is communicated clearly within an organization.

Be clear about "done." Individual lenses and perspectives can create real issues. If you ask the developer if a piece of software is done, they will report it as complete when they have finished writing code. A tester will report it done if they have tested it, had all the bugs fixed, and have certified it. Project managers may call it done when it has been signed off by the Product Owner.

However, the true test of 'done' in the case of the business is; "When can the completed feature be used to provide value to the customer." This means it has to be deployed. Anything short of this is not usable by the partner.

Efforts

Most of the work in the Implementation is in these elements:

- Identify all required environments and document their availability.
    - Quality Assurance (QA) Test
    - End to End (E2E)/User Acceptance Testing (UAT)
    - Performance (Perf)
    - Production (Prod/Live)
- Document connection requirements for each of the environments
- Verify connection to QA
- Verify the authorization process and document it for Intuit systems (this includes documenting the error cases)
- Document the Deauthorization process (i.e. Token Revocation)
- Verify the data delivery, building up the mappings as required for configuration within Intuit systems. (this includes documenting the error cases)
- Document any requirements on the Intuit Platform required to deliver the integration.
- Have a design completed by Architecture and provide requirements and design to the appropriate team for their planning.

Other items are also documented and communicated as part of the process:

- Account types covered by the project
- Are all sets of data already collected by Intuit included in this implementation? Are there new data sets?
- All back end data providers available through the API.
- Key contacts for the project
    - Project Sponsor
    - Project Manager
    - Development Lead
    - QA Lead
    - Support Contact

- How the migration will be deployed.
- Are the customers going to be new users (of Intuit Products) or existing users?
- The implementation support process, both during and post implementation.

Support Phase

Prior to the conclusion of the Implementation, the Partner will be introduced to the Intuit Support Organization to cover methods of requesting technical support, escalation, and additional documentation

Conclusion

API integrations are about providing a seamless solution for shared customers to experience, and the success of the integration should be measured by the customer experience. If you implement an API solution and your customers aren't delighted, you can have any number of downstream problems, costs, and risks.

Every integration partner will see strategic benefits from a structured integration approach.

- Adopt a Standard for your API. Partners integrating with a common standard will have a quicker implementation.
- Create complete documentation for your API and its role in your completed solution. This improves project performance, compliance, and ensures alignment.
- Use a pre-production test environment to completely build and test your solution. This helps ensure your API development will work in the real world with real customers.
- Follow a structured assessment structure such as the Intuit Readiness Assessment. This will keep your integration projects on track and help ensure a successful delivery.

Implementations and API deployments are full of situations where project teams can lose focus, miss targets, and deliver incomplete solutions. All elements of the project team need to be aligned on delivery and quality goals. With the shared customer as the target audience and measure for success, each integration project should reflect a high degree of cooperation. This document has offered some guidance for how to achieve your integration goals.