

THIRD PARTY INTEGRATION

WITH NEOTA LOGIC

2019

INTEGRATING NEOTA APPLICATIONS WITH THIRD PARTY SOFTWARE

Long gone are the days of self-contained software that exists in pure isolation. Instead, it has become natural for innovation departments to think of, and expect, powerful and complex platform rather than trivial and simple programs. Thanks in part to the massive number of offerings from software as a service (SaaS) vendors, such requirements have become more attainable in recent years.

With a vision for empowering subject matter experts to create useful applications via our no-code platform, Neota has aimed to enable integrations with a large number of vendors and tools using our generic Web Services editor. The Web Services editor is a visual component within our larger authoring tool, Studio, which enables integrations with any product that is exposed via a RESTful application programming interface (API); the current de facto standard. Similar to the cURL command, the author in Studio is able to specify a target URL along

with one or more HTTP headers in order to perform a GET, POST, PUT, PATCH or DELETE operation against the target third-party. The URL can be static or dynamically determined via logic at runtime. Support for Basic HTTP and NTLM (Microsoft) authentication is included, so is the ability to create, manage and use OAuth tokens. Responses received from the web service call can be stored in a MIME variable, or parsed into its components when the HTTP response is in XML or JSON format. As usual, when invoking a secure URL (that's the S in HTTPS), the traffic is encrypted across the wire. With access to this powerful basic building block of the RESTful API, multiple web service calls can be combined to accomplish more complex tasks.

Some examples of successful third-party integrations that Neota authors have accomplished include DocuSign, SharePoint, Salesforce, HighQ, Kira, Wolfram, AWS, Google Maps, PowerBI, BigML, iManage

GreenID, TextRazor, Dandelion and Amazon QuickSight.

A Tradeoff of Power vs Usability

"But what about integrations that require half a dozen RESTful calls to complete a useful task, wouldn't a dedicated editor come in handy"? Indeed, hiding RESTful API complexities via simple and elegant syntactic sugar editors is useful for speeding up authoring efforts, and it is possible to supply such editors for some of the market favorites that authors often integrate with. However, if a third-party RESTful API is complex enough to warrant the creation of a dedicated editor, then usually a tradeoff quickly arises when assessing which commonly desired features of the API to expose via the dedicated editor, which features to bundle together, what default values to use for some settings, and which functionalities to entirely omit. Moreover, as API capabilities and their common usage evolve over time, there becomes a need to periodically revisit and evaluate the features exposed via the dedicated Studio editor. In general, we prefer to complete several integrations with a third-party API before defining a useful subset of features that should be exposed via a dedicated editor. The next section offers an alternative to repeating laborious integrations with third-party services.



Neota as a Service (NaaS)

So far, we have discussed invoking third-party software from within Neota's platform but have not discussed invocations in the opposite direction. With Neota's mission being to enable non-programmers to capture their subject matter expertise in the form of an interactive web application that can be run from the browser, embedded as a link or in an iframe within another webpage, etc. That alone is useful but providing an additional no-code mechanism by which applications can be transformed to expose a RESTful API empowers the subject matter expert at a whole new level. The result is headless (non-interactive) applications that can be consumed by systems and servers without human interaction from a browser. This is facilitated in Studio via the API Names editor. Using this feature, an author is able to assign an API name to any variable within the Neota application, as well as a designation of the variable being an input vs an output of the application. As such, simple APIs can be defined by which only a subset of variables are exposed, while the other guts of the application such as logics, internal variables, third-party integrations and other complexities, are completely hidden within the application. All the consumer of this API needs to do is make an HTTP POST request to the application URL and specify in the POST body the input variable values using JSON format. Upon receiving the inputs, the application performs the necessary operations within, and then returns back a JSON response containing the output values. In true black box fashion, the caller need not worry about what takes place behind the scene. Appropriately, we refer to the above as Neota as a Service (NaaS).

There are several benefits to the NaaS paradigm. First, it allows for third-parties to integrate with Neota. Second, it allows for Neota applications to be organized into smaller applications that perform one task, and perform that task really well. We can then for example think of a SharePoint NaaS application, where the complexity of the multiple API calls are all hidden within this black box, and a simpler list of input and output variables are

exposed instead. While this NaaS application can be consumed by third-parties via a cURL command, a NaaS application can also be consumed by other Neota applications. After all, the NaaS API is nothing but a RESTful API, and the Web Service editor in Studio is designed to consume RESTful APIs. One can then envision a graph of NaaS applications daisy chained together to construct more complex structures in a pattern very similar to how several UNIX commands can be piped to perform a more complex task. Sure enough, if the NaaS application is kept up to date with any changes in the SharePoint API, then all of its callers benefit from the change without needing to be altered since the complexities of the third-party APIs are centralized within one application. An arsenal of NaaS applications library will emerge and continue to expand over time. An added benefit is that well-versed non-programmers can maintain NaaS applications without requiring allocation of software engineering resources. Compared to involving engineering resources to create dedicated third-party API integration editors in Studio, creating NaaS applications is in some aspects more flexible and favorable.

In order to further simplify invoking one NaaS application from another Neota application, Studio v9.2.0 (Q1 2019) shipped with a new NaaS editor which is capable of automatically detecting the API exposed by NaaS applications. Simply select a target application from the applications list drop down menu, and the inputs and outputs will be automatically displayed for the author to map the input and output values to variables within the caller application. As such, an existing NaaS application can be consumed from another Neota application in as quickly as a matter of minutes.

While NaaS makes it possible to package third-party integrations into minimum viable products (or rather minimum viable Neota applications), NaaS has its limitations. In particular, NaaS is not suitable for packaging user-interactive third-party APIs. For example, a typical integration with DocuSign has a user starting within a Neota application in the browser, then the user is taken to a DocuSign page to complete the signature process, and the user is finally send back to the Neota application. While the non-interactive portions of the DocuSign integration can indeed be bundled within a



NaaS application, the interactive portion does not fit well into the headless JSON input / JSON output web service paradigm. Instead, what is needed is a notion of an application redirect so that the user remains in the browser for the actual signing experience.

Payment Gateways

We now make a brief detour to discuss a different flavor of integrations. Suppose an author using Studio has a need to stand up a paywall (payment form) on the second webpage of a five-page application. Alternatively, suppose the author prefers to allow user input to proceed free of charge until the very last webpage of the application, and that the resultant report is only made accessible for download or is emailed upon collection of a fee. There is a clear need for a simplified paywall integration.

Due to special circumstances, a few of Neota's clients have integrated with third-party paywall solutions via explicit invocations of RESTful APIs from the Web Services editor. While this is possible, our preference and official support is for collecting payments via Stripe (similar to a PayPal account). We have provided a simple mechanism by which a Stripe payment account can be linked to a Neota application, and the author of the application is then able to simply add a payment logic in Studio. The payment editor takes a handful of inputs, including the amount and currency of the charge, which can be of fixed value or dynamically computed based on some logical criteria in the application. Dynamically computing the cost is, at times, useful. For example, an application that runs in both simple and advanced mode may have different pricing for each mode. Certainly, as

with any other logic defined in Studio, the author has control over the conditions that must be met before the paywall is presented in the Neota application when run in the browser.

At runtime, the credit card information collected from the end user is passed directly from the browser to the Stripe servers without touching Neota's server, thus keeping the PCI DSS compliance requirements at minimum. Users can opt to save a card on file (again, the actual card information is sent to Stripe and not stored directly in any Neota system). Payment history is made accessible to the Neota site admin from within our portal (Workbench). Most international currencies are supported, and all major cards are accepted.

Preferred Integrations

While providing native support for certain functionalities directly within Neota's platform is sensible, clients often have their preferred third-party tools that they would rather integrate with; sometimes requiring several third-party integrations from within a single web application in order to accomplish a useful task. In addition to its powerful reasoning engine, Neota's platform can serve as a viable backbone or glue that brings it all together into a single web application.

CONTACT US

info@neotalogic.com