Service-Oriented Architecture

SRP FrameWorks and the SRP HTTP Framework module both use a Service-Oriented Architecture (SOA) approach to assist with general programming and business solution tasks. Each stand-alone portion of logic is referred to as an *application service*. To help with organization, services with a common general purpose are grouped into *service modules* (also known as *service handlers*). In OpenInsight, *service modules* are represented by stored procedures: HTTP_Services, `Security_Services`, etc. Within each *service module* are the respective *application services* (or just *services* for short). *Services* are organized within *service modules* via *label* and *return* blocks.

Our SOA paradigm fits within the broader architecture known as Model-View-Controller (MVC). As previously documented, the *Model* represents the database and business logic processes. The *View* represents the user interface (e.g., OpenInsight Form, HTML) and its respective event logic (e.g., BASIC+ Commuter Module, JavaScript). Finally, the *Controller* represents the middleware that allows the *Model* and *View* to communicate with one another. *Service Modules* and their respective *Services* typically fall within the *Model* layer. Consequently, *services* should be designed carefully to avoid actions that belong to a *View*. For instance, this means a *service* designed for business logic should not invoke a UI. Likewise, *View* logic should not directly access the database. This requires discipline and an acceptance of the MVC philosophy (of course, the inherent design of any web application enforces this architecture already).

Well designed services are more than individual pieces of programming logic. Here are some of the key principles of a service oriented approach:

- **Reusable business processes** - Logic that tends to be called more than once or from more than one location should be wrapped up in a service.
- **Self-contained** - Services should be capable of performing their duties only using the supplied arguments. Thus, data stored in global variables, commons, etc., cannot be used, since this establishes a state that might not be duplicated under other conditions. In this regard, services are *black boxes*, engines that receive incoming data and send back a response. The same incoming data will always yield the same response.
- **Modular** - A service can rely upon other services, as long as each service follows the same key principles described in this article.

When a service is designed using the above principles, it is also suitable for unit testing. This makes it easier to validate the proper functionality of any service without having to run the full application. It also makes it easier to confirm that changes to the service logic (such as refactoring) does not alter expected behavior.