Designing Payment Processing Applications with Flux

Customers are streamlining their payments processing with Flux. Are you?

Reading Time: 5 minutes

Many came to using Flux originally for its job scheduling capabilities. Yet it's surprising to say a significant group of Flux customers make minimum use of Flux's sophisticated time expressions, business calendar support, and powerful scheduling. These customers often are drawn to Flux for its file orchestration capabilities, in particular when it comes to processing payment files.

Developing financial applications is hard work. Using Flux streamlines the development process, but using Flux is not simply 'adding a little workflow' into your application. Flux is a platform as opposed to an application or library, and as such it requires a shift in design perspective to effectively exploit its capabilities and strength. Effective design and architecture is key.

Many forms of payment file processing follow fairly straightforward paths of execution. Files are received, parsed, validated, reviewed, corrected, balanced, reformatted for submission to legacy posting and settlement systems. These steps are fairly common, regardless if the file contains, for example, ATM transactions, check payments, or other payment data.

In payment processing, files are generally first-class and very visible objects

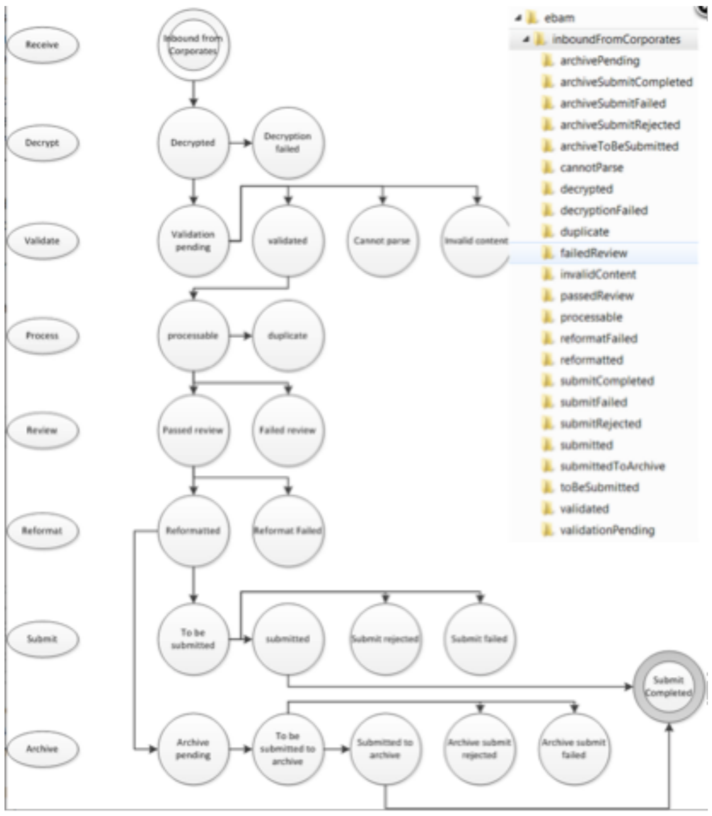
In payment processing, files are generally first-class and very visible objects within the processing – they are not simply a container for any collection of payments. Often these files have header and trailer records, and contain information organized into batches, blocks, embedded lists, or some other structures. The definition of these files is often governed by ANSI, ISO, or a vendor's proprietary standard. Significant and often memory and CPU intense file validation processes may need to be applied throughout the processing of these files. It is not uncommon to see such files be processed against large software components providing fraud detection, duplicate payment detection, and image quality assurance within the first few steps of its processing.

One can design such applications from a variety of perspectives such as a data-driven approach, an object-oriented approach, as well as other design perspectives. Many Flux customers take a very simple, yet quite pragmatic approach that shares attributes with rapid application design (RAD) techniques. They map the states of the file or messages to folders on the file system. In point of fact, many applications actually deploy using this model – although others simply use this as a design ‘thought exercise’ and eventually store the file and its interim states into databases or message queues.

Flux followed this approach in its design of Flux eBAM for Banks

Flux itself followed this approach in its design of Flux eBAM for Banks. While not a payment processing system per se, Flux eBAM for Banks shares many attributes with such systems. eBAM (electronic bank account management) utilizes the ISO 20022 XML standard for relaying information about a corporation’s bank accounts between its own corporate systems (e.g., ERP, HR, cash management) and its banking partners. The standard defines 15 message types, include bank account opening messages, bank account closing messages, and account reporting messages. Messages can be sent and received via my number of channels, including the SWIFT network, a bank’s dedicated network, even email. Once received, each message can be treated as a file, and the various states of the file are defined as directories.

The following diagram draft, developed early in the development process, illustrates this model. On the left is the process performed, and In the middle are the candidate states of the



message after the processing is completed. The hierarchical list on the right is a directory tree for a prototype deployment onto a file system.

Once defined in this manner, Flux's file orchestration engine was utilized to implement this model. A workflow model was constructed to implement the above processing model. Each process is defined in the workflow as an action, some actions being provided as off-the-shelf actions, others being custom Java actions. Each action takes as input the message received from a triggering event defined in the workflow (e.g., file exists triggers or database trigger). The action then processes the message, in the particular state shown above, into the specified target state. The resultant state of the message is placed in the destination directory, queue, or database. In some cases the actual file contents may change (e.g., in the 'Reformat' process shown above). In other cases the message contents are simply moved – as when a file moves from 'Validation pending' to 'validated' since the validation process simply assures the format and content of the file, without altering its content in any way.

This model of processing can be flexed easily to address multi-tenant applications where segregation of client data must be maintained. Creating many instances of these directory or queue structures by client is straightforward with minimal impact to the processing model. Using dependency injection can provide further improvement to the flexibility and ease of deployment for a variety of targeted environments.

Now the true power of file orchestration comes into play

After developing the application in such a manner, the true power of file orchestration comes into play. The Flux file orchestration engine can be configured and tuned, without any coding, to match the application to the real world processing challenges it will encounter. For example, pinning and load balancing are two key features of the Flux engine. Some processes may be very CPU intense and long-lived, others consume a large amount of memory. Large payment files may take many minutes to reformat, and if too many files are engaged in such a process the application may stall for lack of resources. Being able to 'pin' such processes to specific machines, and 'throttle' their execution to a specified limit can mitigate such circumstances. Load balancing work across processing nodes is important in ensuring effective usage of all available processing resources, especially under the high load volumes seen in payment processing systems.

Payment processing occurs within legal mandates of compliance and audit

Payment processing occurs within legal mandates of compliance and audit. Within Flux, every action is audited. Every workflow's and action's start and stop time are logged. All exceptional events are recorded. Methods are provided to scan the audit log and trigger other workflows and send notifications in the event compliance, audit, or processing deviations are detected.

Payment processing also is generally mission-critical to the institution. Knowing exactly where files are in their processing is essential. Flux provides monitoring and control features via its operations console for IT operations personnel, and a complete REST and Java API for integration with existing enterprise monitoring tools.

In summary, Flux's file orchestration capabilities provide much in the way to expedite and facilitate flexible, robust, and maintainable payment processing applications. Many payment processing applications running today embedded Flux to leverage these capabilities to their enterprises's advantage. Have you?

About Flux

Built on the 13 year foundation provided by Flux software platform, Flux provides Electronic Bank Account Management (eBAM) solutions for banks. Electronic bank account management replaces slow paper-based processes with electronic efficiencies, reducing human errors and providing greater transparency into bank and corporate operations.

Banks that offer an eBAM solution possess a critical market advantage in their efforts to expand and retain their corporate customer base.

The Flux software platform orchestrates file transfers and batch processing workflows for banking and finance. First released in 2000, Flux has grown into a financial platform that the largest US, UK, and Canadian banks and financial services organizations rely on daily for their mission critical financial systems.

Contact Flux

+1 702-789-0907
sales@flux.ly
www.flux.ly