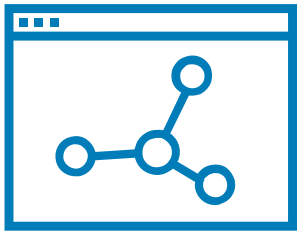

Administration Training
for Developers
Boomi

Admin Training Agenda

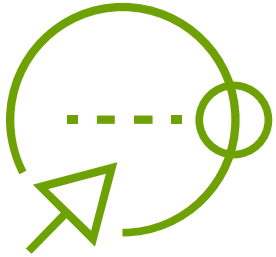
- Administration Overview
- Process Deployment
- Process Automation
- Connection Licenses
- Process Reporting
- Process Troubleshooting
- User Alert Setup
- Process Deactivation

How It Works



Build

Using the AtomSphere library of connectors and maps, visually design your integration processes and load them into a lightweight, dynamic run-time engine called an “Atom” for execution.



Deploy

Deploy your Atoms to the AtomSphere for SaaS, PaaS, or Cloud integration or safely behind your firewall for on-premise applications.

Manage

Monitor and maintain the status of all your deployed Atoms, integration processes and trading partners, regardless of location, using a feature-rich web-based dashboard.

SaaS Training Support Scenario

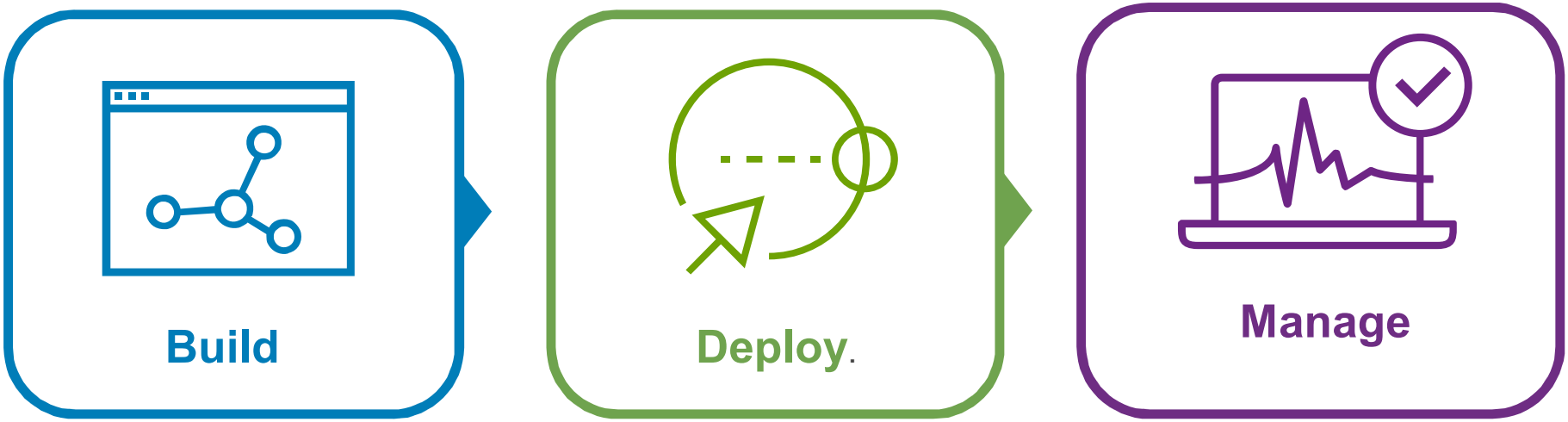
Business Use Case:

- Sales team enters prospect information into salesforce.com
- Operations team wants to query Salesforce Accounts (Prospects) and send responses to an Organization Tracking System for lead management
- Support wants to monitor the integration to ensure strategic IT success

Administration Goals

Prospect Tracking

- Automate the Process to query newly modified accounts
- Deploy to an Environment and monitor live transactions
- Troubleshoot and retry failed executions and documents
- Develop and release Process updates for advanced logging
- Enable administration features for advanced alerting



Build Tab (Test Mode) vs. Deployed Processes

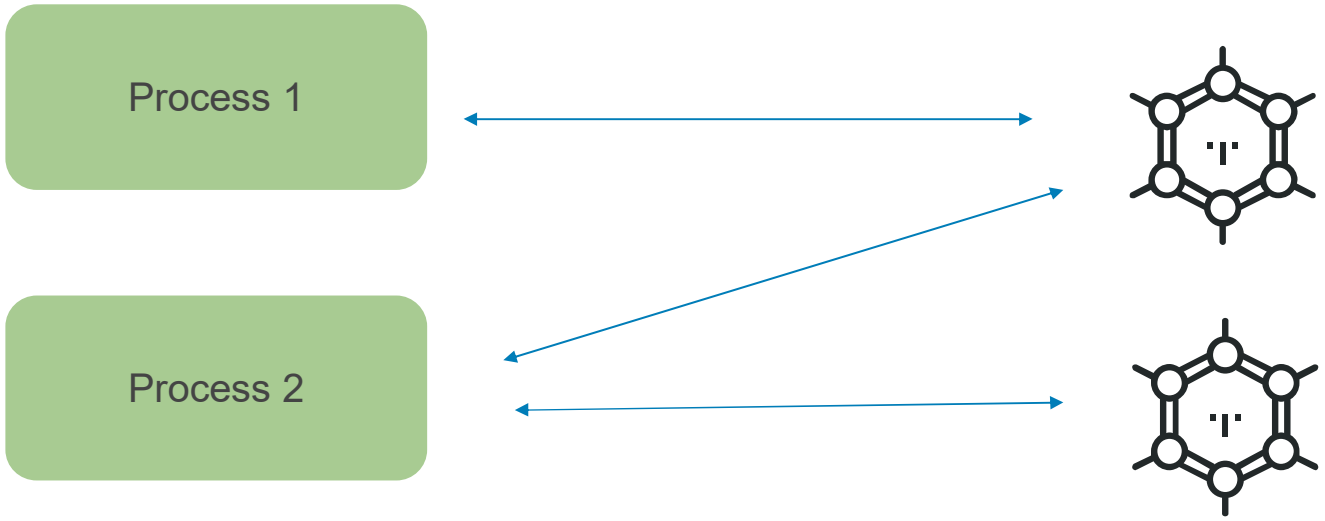
Test Mode supports unit testing of a limited set of documents

- Connectors only support retrieval of:
  - 100 – Max Document Count
  - 10 MB – Total Data Size

Deployed Processes support strategic monitoring and unlimited data

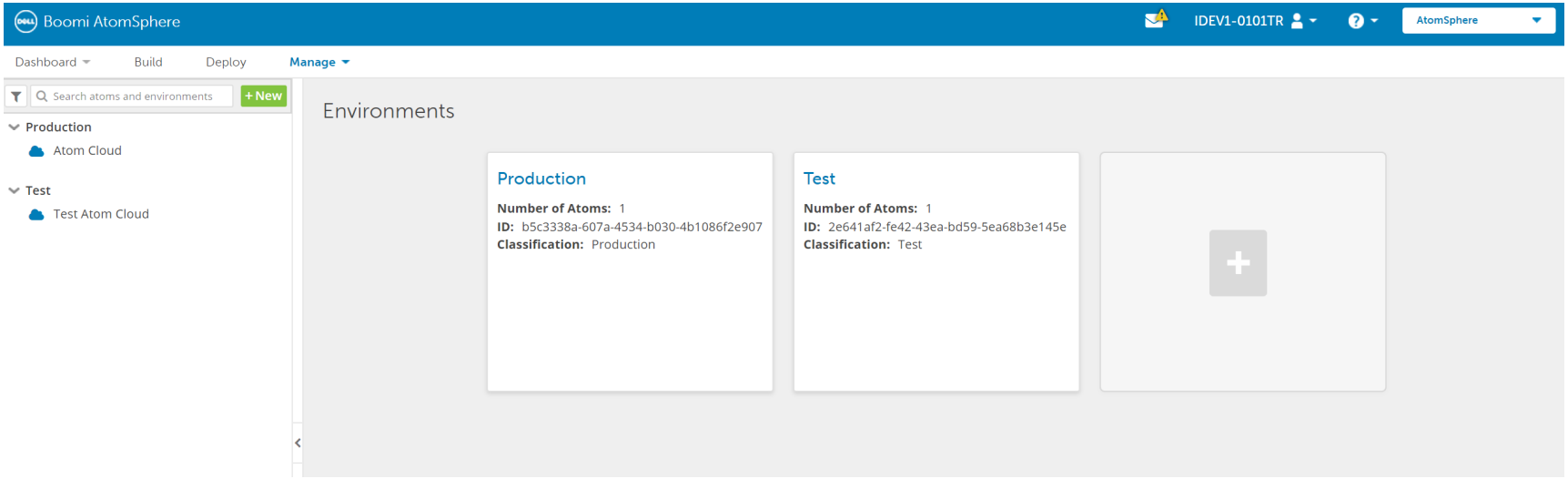
- Execution and Document History searching
- External Alerting (Email & RSS)

Process Deployment



Environments

- Accessible via Manage→Atom Management
- No limit to the number of Environments
- An Atom can only be assigned to one Environment



Walkthrough

Walkthrough:

- **Exercise 1: Check configuration of Production and Test environments**
- **Exercise 2: Deploy the process**

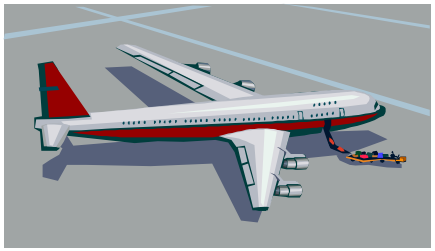
Page(s): 5-10

Deploy vs Execute

- Deploying a process does NOT execute the process into an active production state
- Once a Process is Deployed, it needs to be executed either by:
  - Manual execution

    or
  - Scheduled execution

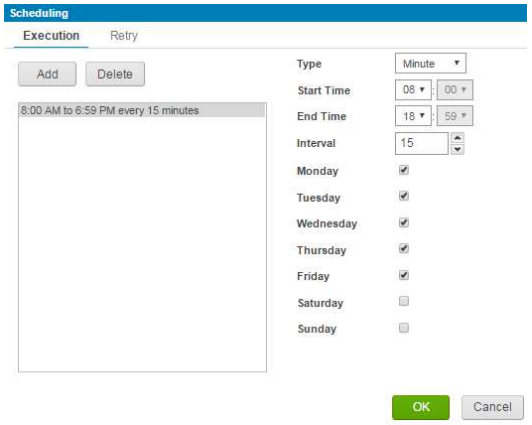
Deployed

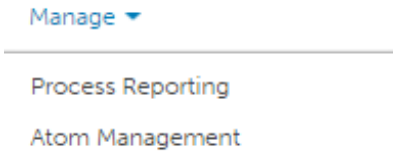


Executed (Manual or Schedule)

Process Automation

One Minute Intervals





Process Automation

Internal Scheduling

- Schedule indicates when process is set to run
- Create schedules to intervals of 1-minute
- Create *advanced* schedules to execute on specific minutes
- Retry schedule indicates when failed documents are retried

Real Time Integration

- Supports event-driven integration
  - Web Service Publishing (HTTP/SOAP)
  - AS2 Shared Server
- Creates new execution thread for each push from a client application

Walkthrough

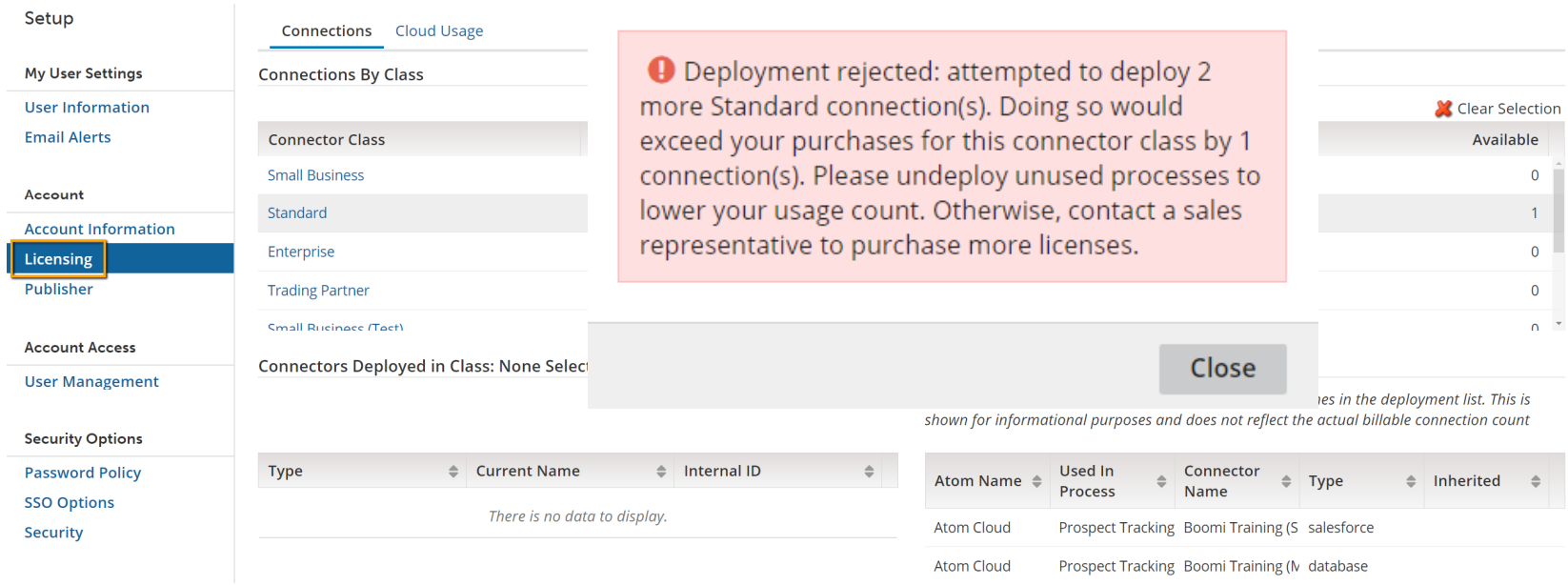
Walkthrough:

Exercise 3: Configure a process schedule

Page(s): 11-12

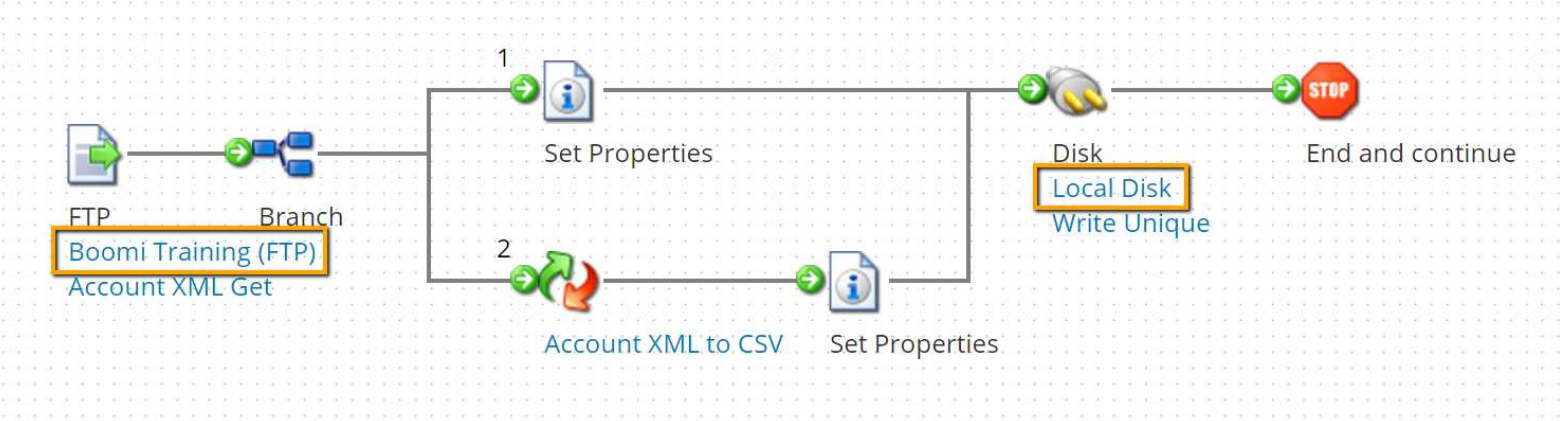
Connection Licensing

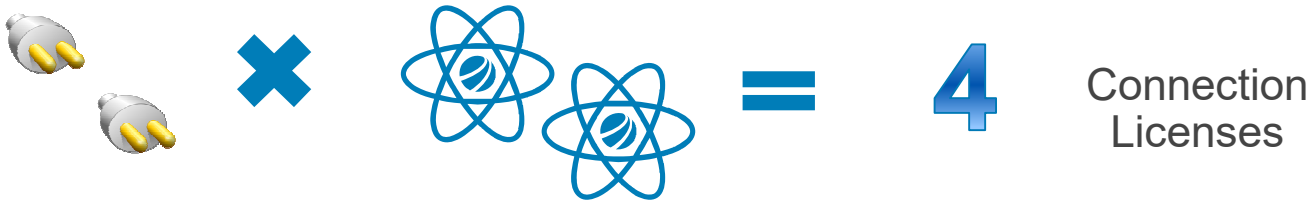
- Under Setup Menu >> Licensing
- Identifies connector deployments across processes and atoms
- Limits process deployments per account



Connection Licensing

- A process contains a FTP and Disk Connector
- An AtomSphere user deploys the process to **2** different atoms:





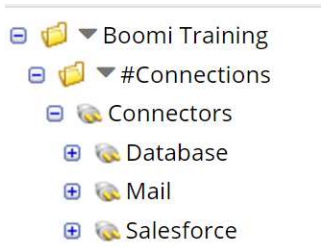
4 Connection Licenses

Connection Licensing: Best Practices

- Avoid using duplicate Connection components in a single process
- Create a shared connections folder to house common integration components
- Re-use common components throughout all integration projects
- Work with Dell Boomi Account or Partner Manager to define strategic license count

Connections By Class

Connector Class	Purchased	Deployed
Small Business	0	0
Standard	3	2
Enterprise	0	0
Trading Partner	0	0
Small Business (Test)	0	0
Standard (Test)	3	0
Enterprise (Test)	0	0



Walkthrough

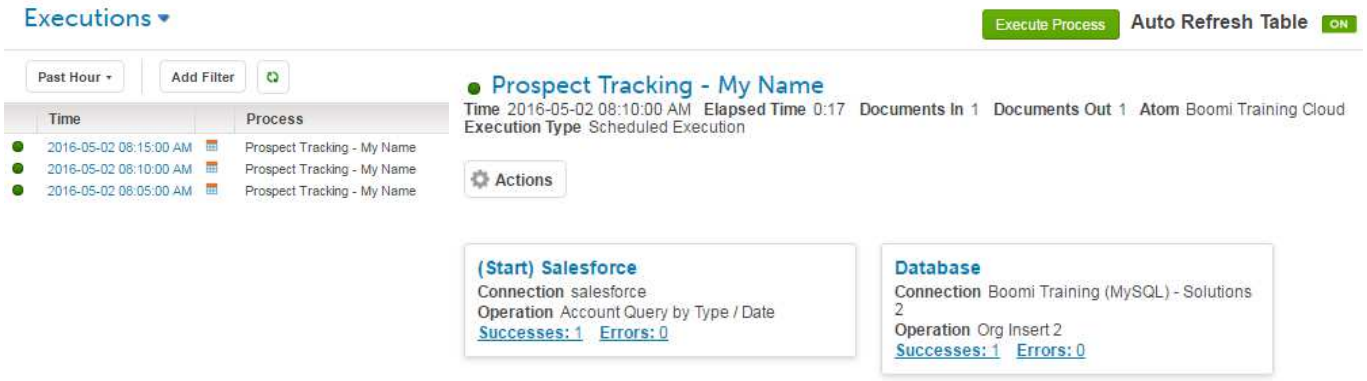
Walkthrough:

Exercise 4: Review the deployed connection licenses

Page(s): 13-14

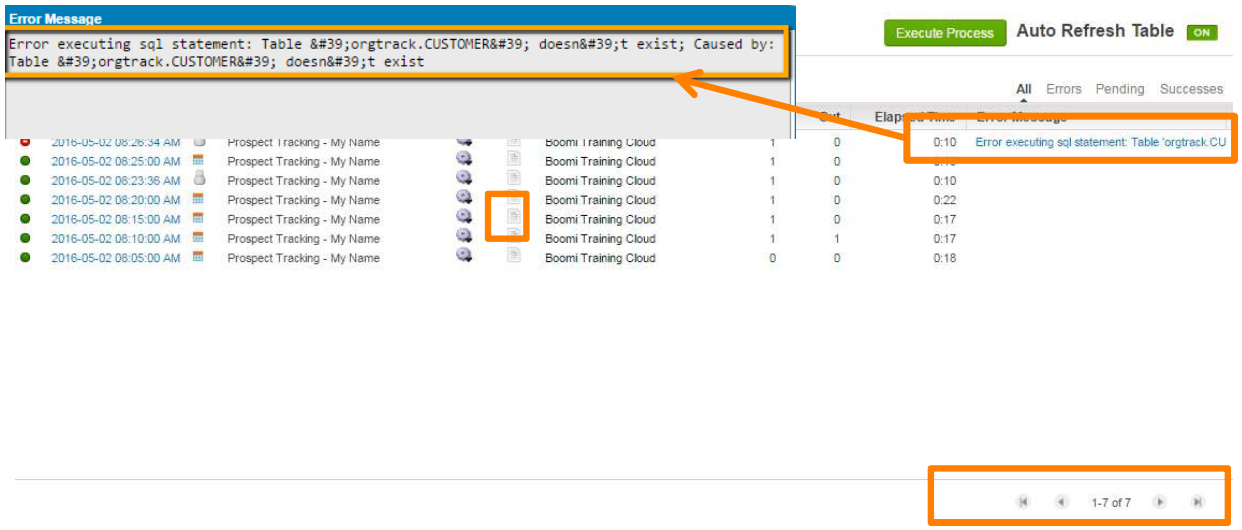
Process Reporting

- Defaults to Executions launched in Past Hour across All Atoms
- Set search filters and Refresh Search
- Highlight execution records to display document activity
- Manually ‘play’ deployed Processes in specific Atoms



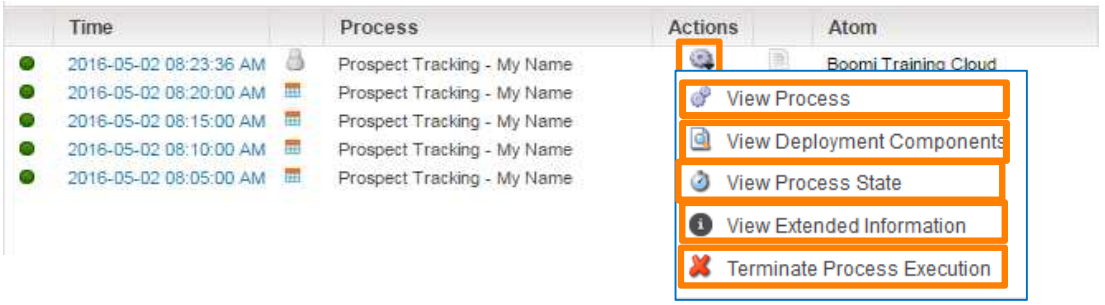
Process Execution Statistics

- Click on filter links to sort by Execution Status
- Page through sets of 25 execution records
- View high-level Process log
- Double-click to view details of the first Error Message returned:

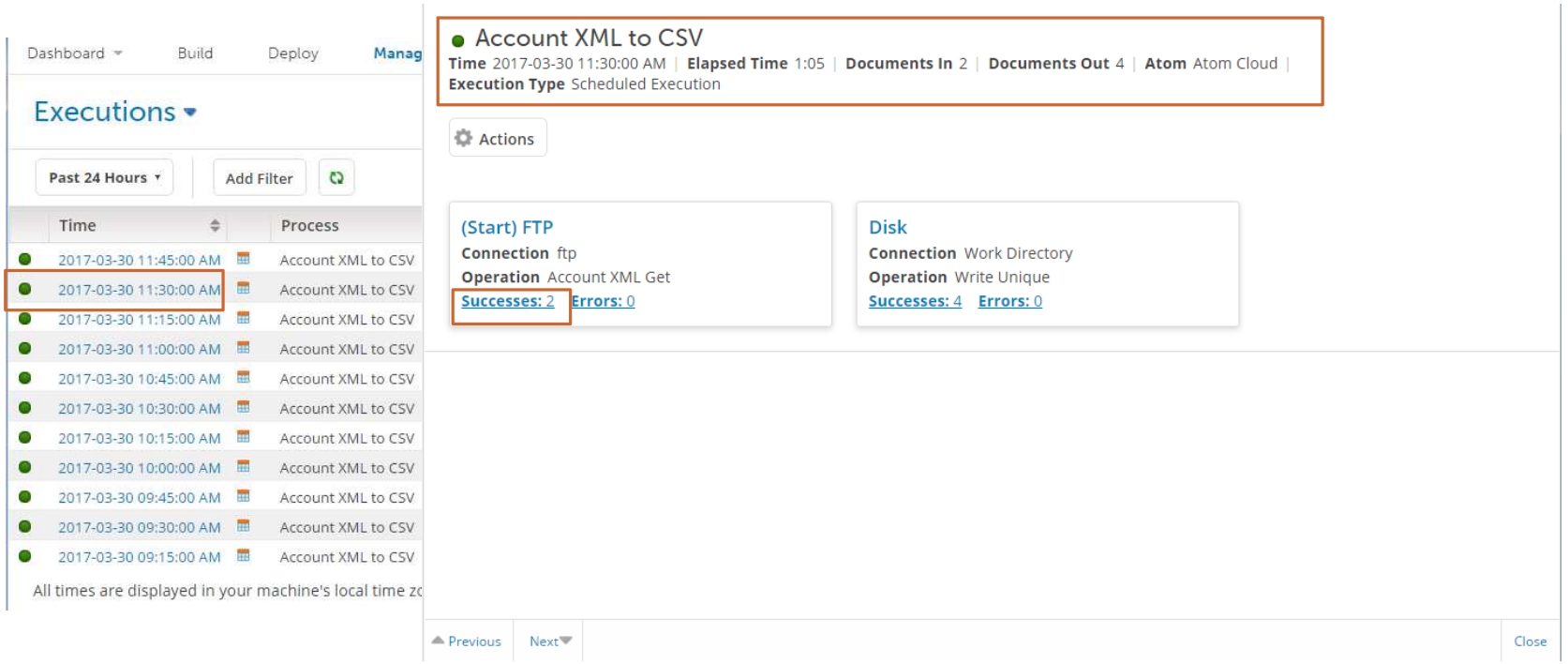


Process Execution Actions

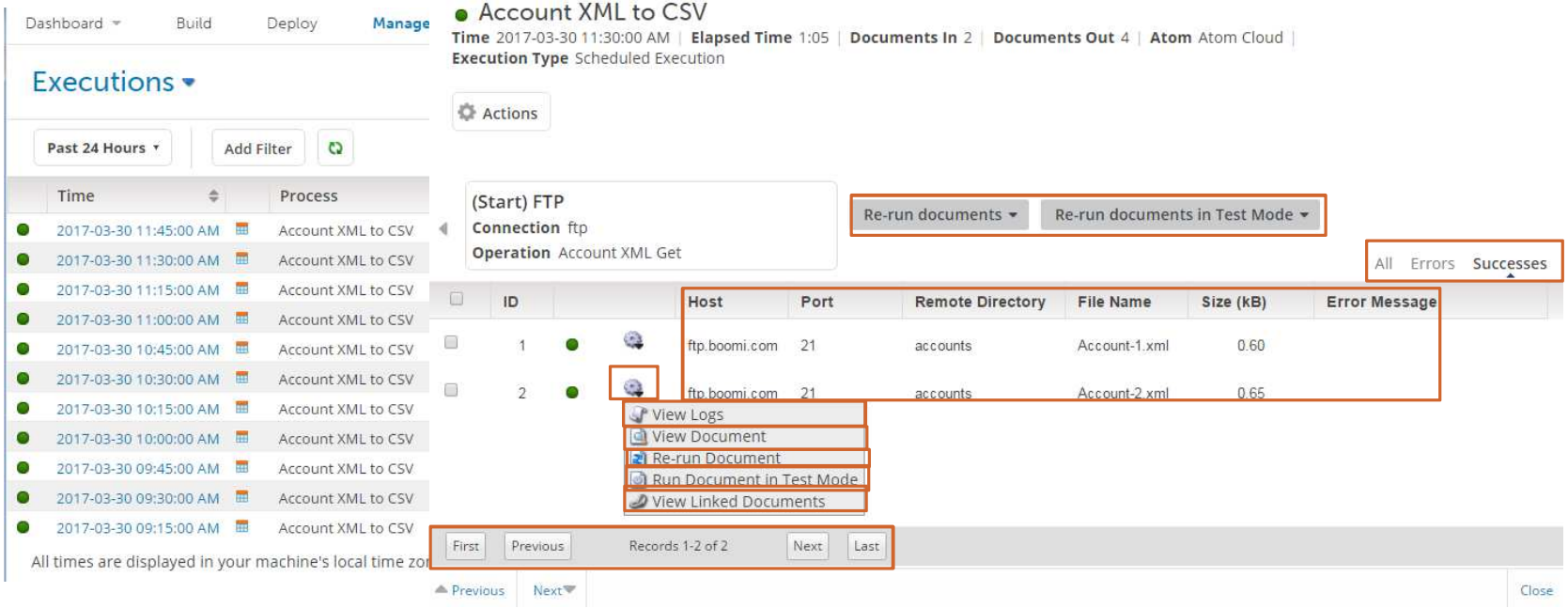
- View Process
  - Open the Process on the Build Tab
- View Deployment Components
  - View and open component versions on the Build Tab
- View Process State
  - View real-time information about step execution and duration
- View Extended Information
  - View IDs and Download Execution Artifacts (Admin Only)
- Cancel Process Execution
  - Terminate Processes in a Pending state



Document Statistics



Document Statistics



Walkthrough

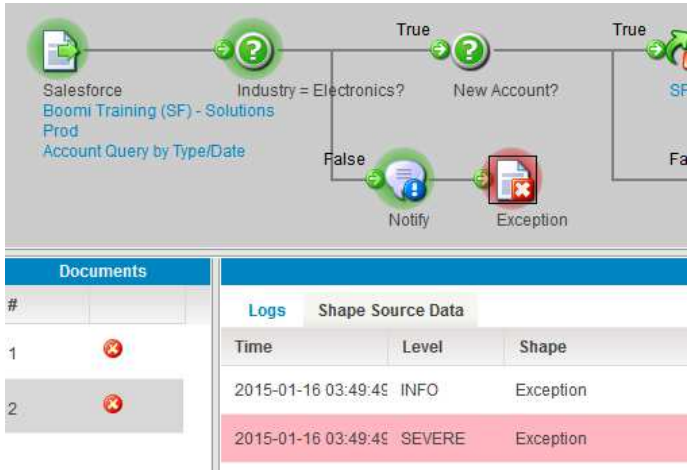
Walkthrough:

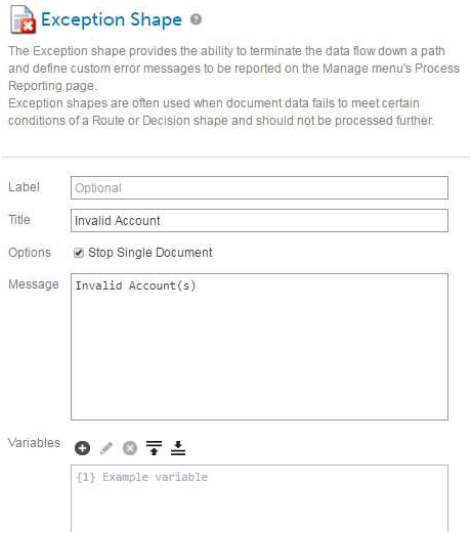
- **Exercise 5: Track live executions in process reporting**
- **Exercise 6: Review live document information**

Page(s): 15-19

What is the Exception Shape?

- Terminates document(s) in a Process flow
- Flags the Process Execution as ‘Failed’
- Logs custom error(s) in the Process and Document logs





User Alert Example: PROCESS.EXECUTION

PROCESS.EXECUTION - ERROR: Process Admin 1 - Exception Shape Example in Atom Atom Cloud now has a status of ERROR, recorded at 04/28/2017 12:04:58 PM EDT

Process Name: Admin 1 - Exception Shape Example
Execution Id: execution-6f259af5-34e2-4b25-aa1c-b97860dc72dd-2017.04.28
Environment: Production
Classification: Production
Started At: 04/28/2017 12:04:58 PM EDT
Ended At: 04/28/2017 12:05:14 PM EDT
Final Status: ERROR
Error Message:

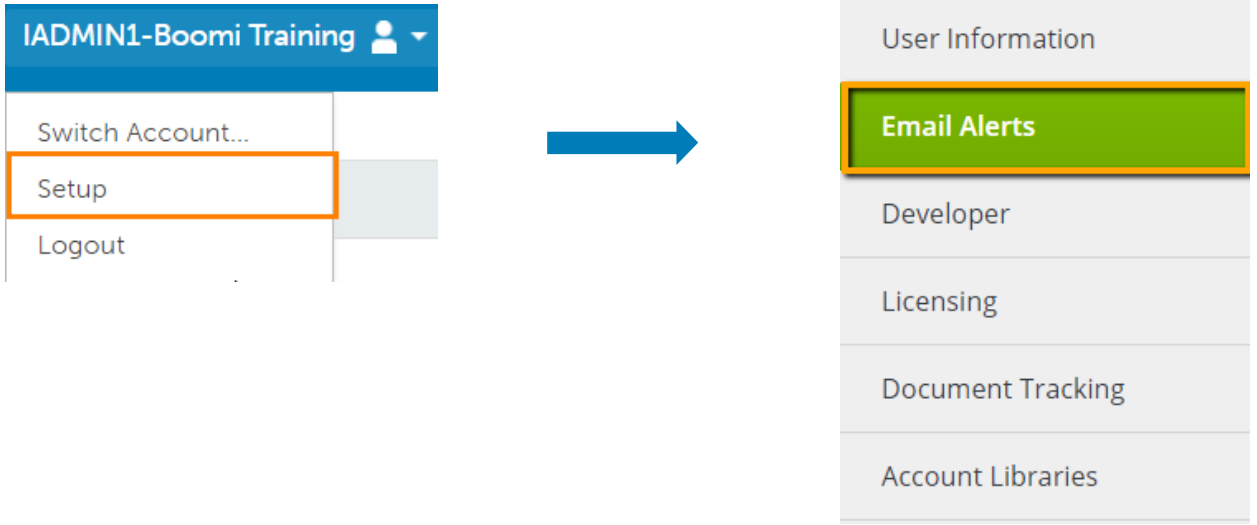
Process terminating -- some documents reached an exception which is set to halt all documents: Exception Shape: This Account BockelCorp has Technology as the Industry

Error Type: DOCUMENT
Errored Step Label:
Errored Step Type: Exception
Error Document Count: 2
Inbound Document Count: 2
Outbound Document Count: 2

AtomSphere User Alerts

Every 5 minutes

AtomSphere Email Alerts



Instructor to Demonstrate

Instructor To Demonstrate:

Exercise 7: Configure a forced exception notice

Page(s): 20-23

Participants to Complete

Participants to Complete:

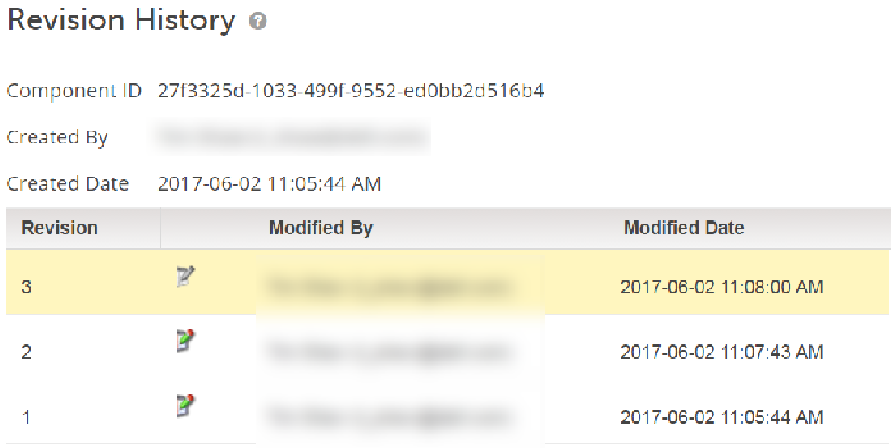
Exercise 7: Configure a forced exception notice

Page(s): 20-23

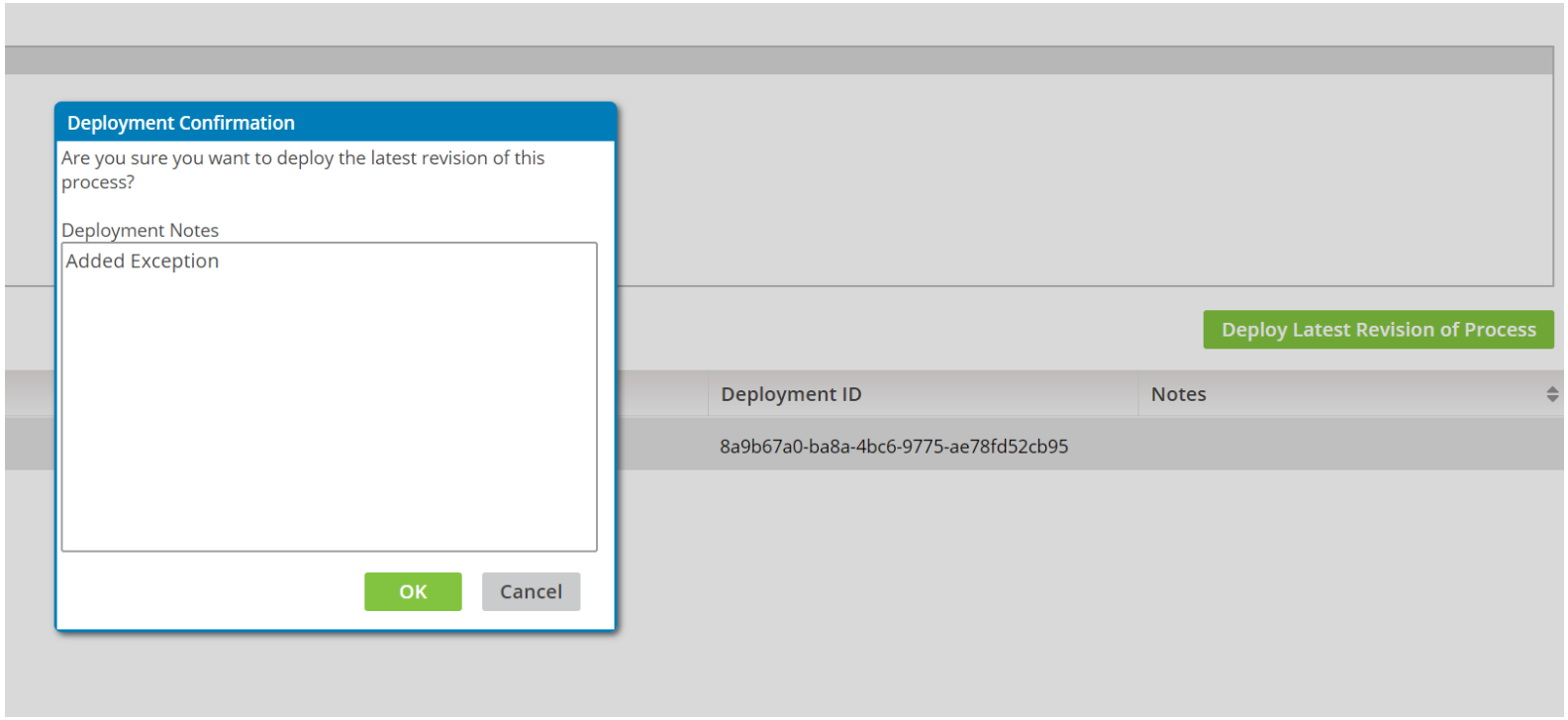
Revision History

- Available feature for all components
- View/Edit versions of a component
- Overwrite invalid revisions (Undo)
- **Each** component has an **independent** revision history

Located in bottom-right below the Process Canvas



Deploy Latest Revision of Process



Walkthrough

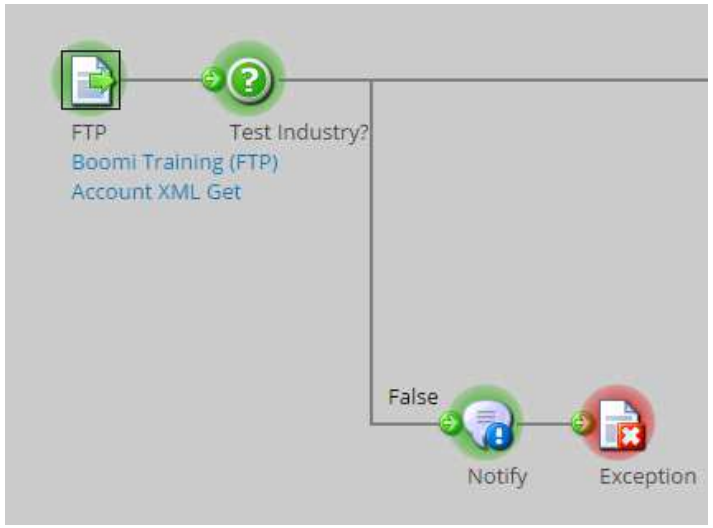
Walkthrough:

- **Exercise 8: Review the revision history**
- **Exercise 9: Redeploy the process and view alerts in process reporting**

Page(s): 24-27

Notify Shape

- Build custom notification messages to subscribed email alerts or RSS feeds
- Can create one unique log per document
- Does not affect document data



User Alert Example - USER.NOTIFICATION

PROCESS.EXECUTION - ERROR: Process Admin 1 - Notify Shape Example in Atom Atom Cloud now has a status of ERROR, recorded at 05/03/2017 8:40:38 AM EDT

Process Name: Admin 1 - Notify Shape Example
Execution Id: execution-de2e2baf-70ba-41de-88a8-2e2de343cae1-2017.05.03
Environment: Production
Classification: Production
Started At: 05/03/2017 8:40:38 AM EDT
Ended At: 05/03/2017 8:41:00 AM EDT
Final Status: ERROR
Error Message:

Exception Shape: BockelCorp has Technology as the Industry

Error Type: DOCUMENT
Errored Step Label:
Errored Step Type: Exception
Error Document Count: 1
Inbound Document Count: 2
Outbound Document Count: 2

USER.NOTIFICATION - ERROR: Notification recorded at 05/03/2017 8:41:00 AM EDT

Title: Invalid Company
Message: Notify Shape: BockelCorp has Technology as the Industry
Environment: Production
Classification: Production

Instructor to Demonstrate

Instructor To Demonstrate:

Exercise 10: Configure a notify shape

Page(s): 28-32

Participants to Complete

Participants to Complete:

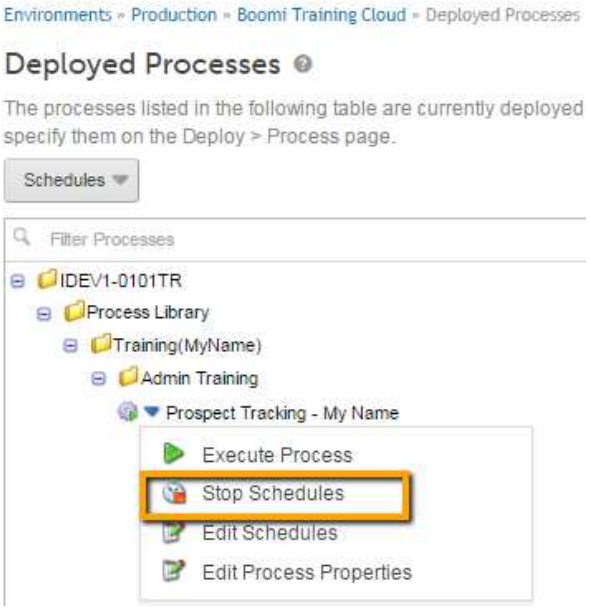
Exercise 10: Configure a notify shape

Page(s): 28-32

Process Deactivation

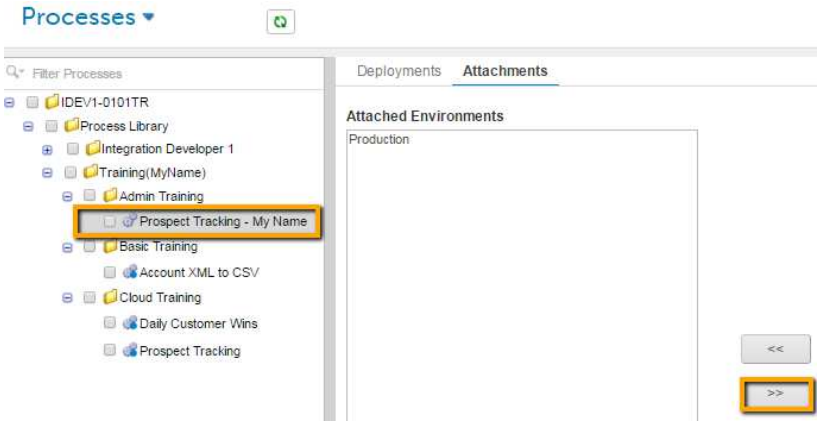
Stop the Process Schedule

Manage >> Atom Management >> Atom >> Deployed Processes>>>Stop Schedules



Detach the Atom

Deploy >> Process >> Attachments >> Unattached

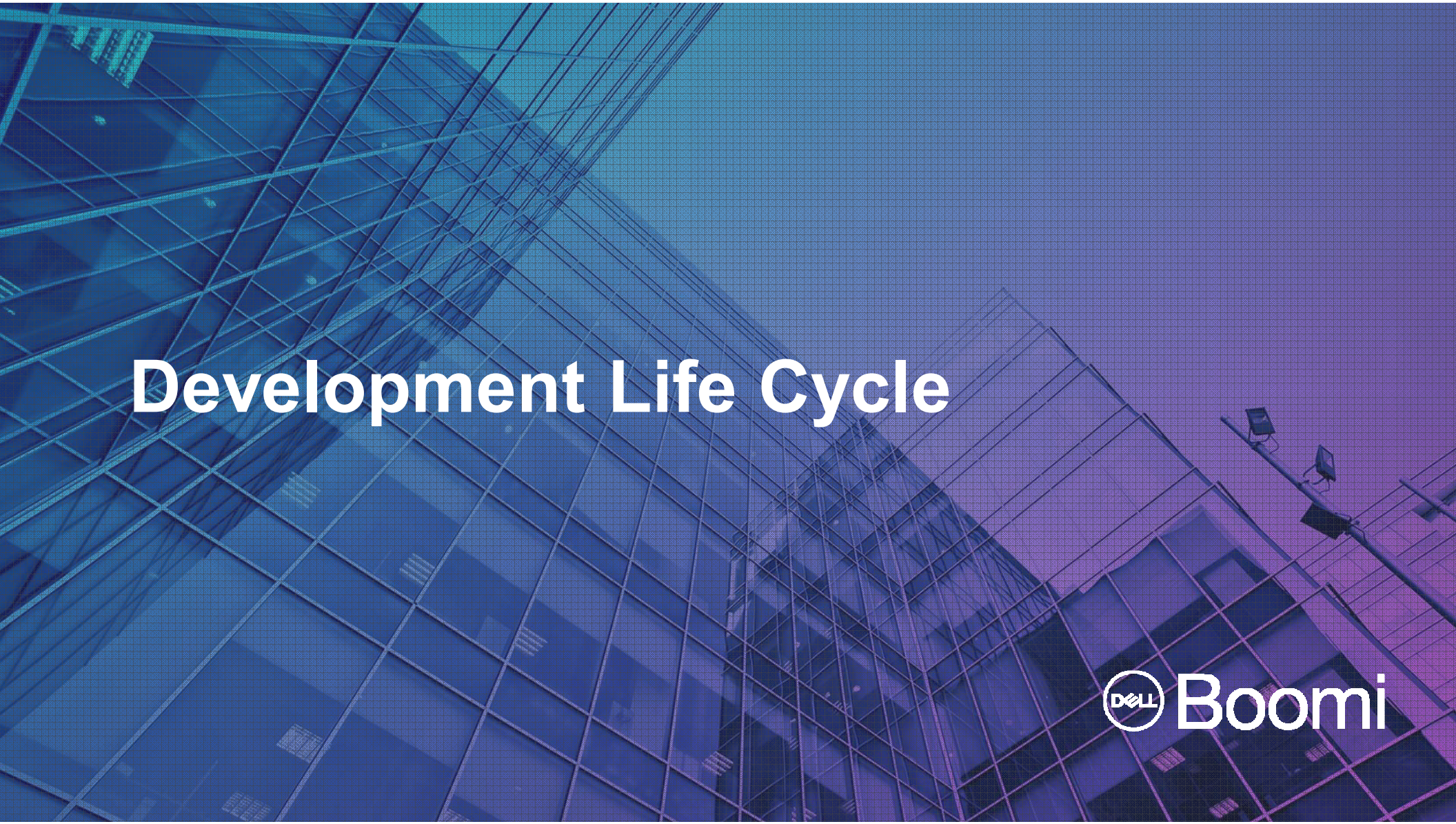


Walkthrough

Walkthrough:

Exercise 11: Stop process schedule and detach environment

Page(s): 33-34


Development Life Cycle
Boomi

What's the Big Idea?

- How to develop and perform change management for integration projects:
  - Development >> Test >> Production
- Review the fundamental AtomSphere concepts and combination of features that enable change management
- Review several common change management scenarios

Importance of the Development Life Cycle

- Ensures that your processes meet the business requirements for the integration.
- Supports the ongoing viability of deployed processes.
- Provides a structure for defining, creating, testing, implementing, and maintaining business solutions using Boomi processes.
- Ongoing and can be adjusted as the needs of the business change.
- The Boomi AtomSphere development life cycle is new and different for both software developers and business analysts.

How Is Boomi Different?

- An AtomSphere process is a document processing pipeline made up of a set of shapes and properties.
- Some shapes and properties are part of the process object and are managed and versioned along with the process.
- Other shapes are separate objects and are versioned separately and can be reused (referenced in other processes).
- Configure a complex integration process on a palette as a series of steps using the simple shapes provided.
- Graphical in nature:
  - Code techniques do not work
  - Source control is “built in”
  - No branch/merge concepts
  - Linear development
  - Everyone works on the latest version, no “check in / check out”

Development Life Cycle

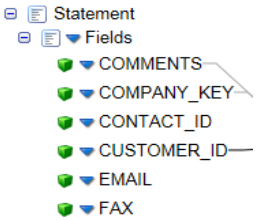
- **Build Components**

What are Components?

- **Components** are the building blocks of integrations: Processes, Connections, Operations, Maps, Profiles, etc.
- Components promote reusability: connections, “utility” functions.
- Components can be referenced in any number of processes (or other components).
- Although components are often created within the context of a process, they exist and are maintained independently.
- Configuration is “by reference.”
  - If two processes reference the same map and you change that map within one process, the other process is affected by the change.
  - If you remove a component from a process, it still exists in the Component Explorer; it is NOT deleted.

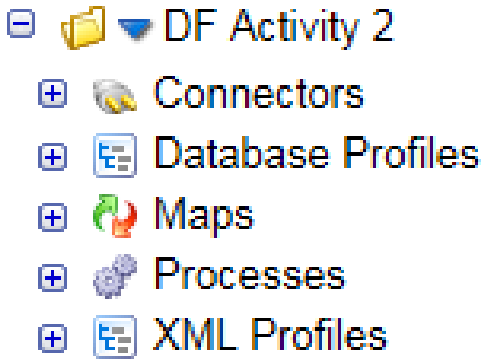
Versioned Components

- Process – “executable” component that references other sub-processes and components
- Connector – contains a Connection and an Operation
- Profile – the “metadata” that describes the structure of a document being processed (a record layout)
- Map – set of logic and mappings that transform data from a source Profile to a destination Profile
- User-defined function – a reusable set of scripts or functions linked together to perform complex logic



A Hierarchy of Objects

- Process is a hierarchy of objects, separately edited and versioned.
- Test runs use the latest version of the process, its attributes, and all sub-objects.
- Sub-objects include:
  - Maps
  - Profiles
  - Connections
  - Connector Operations
  - Functions
  - Non-dynamic Process Properties



Development Life Cycle

- **Reusable Components**

Why Is Reuse Important?

- Reduces duplication of effort, share useful functions and features, and manage common functions in one place.
- Additionally, this helps you to effectively use AtomSphere licensed connections.
- Components to consider for reuse:
  - Connections – to stay within licensing limits
  - User-defined functions – useful for sharing a common transformation in a Map
  - Processes – often a common sub-process is used to establish authorizations, get a token, or start a session at an end-point
  - Cross Reference components
  - Process Properties

Reusing Connections

Considerations:

- Boomi AtomSphere licensing is “per connection”
- One license per connection per atom deployed to two separate deployed Connection objects to the same endpoint requires TWO licenses
- It is essential to keep control of your connection components and ensure reuse by all processes that will be deployed
- A recommended best practice is to enforce folder security on folders containing shared components

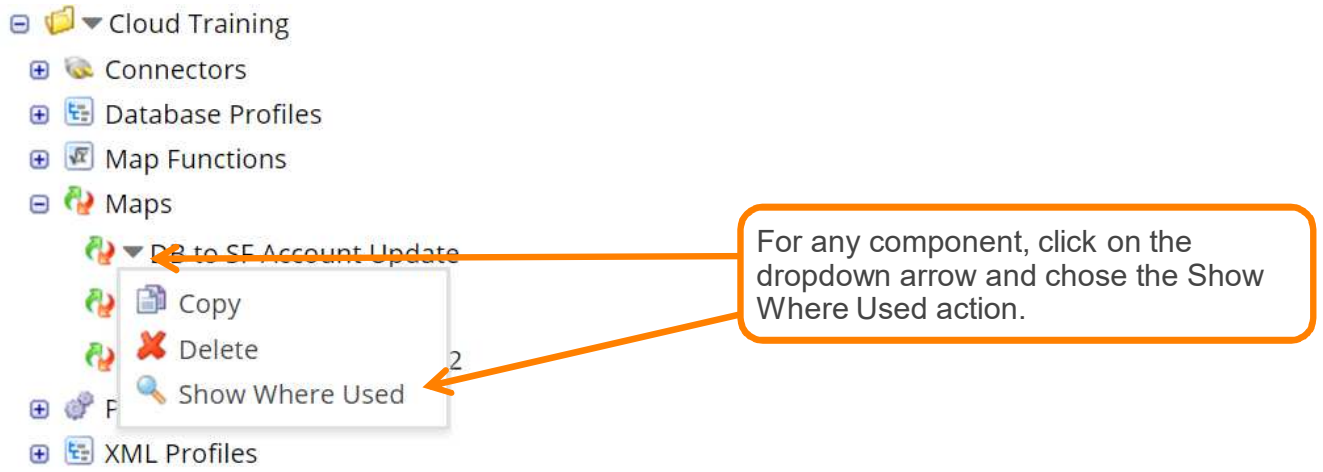
Reusing Connections

Guidelines:

- Maintain a single copy of each Connection object to be deployed
- Keep these in a common folder – a folder named #Connections under the root is convenient
- Any process reference to a connection uses these
- Do not store credential information on the connection
  - Use extension values at the environment level
  - Set credential information on connections to dummy values
- Immediately delete duplicates created by Deep Copy operations

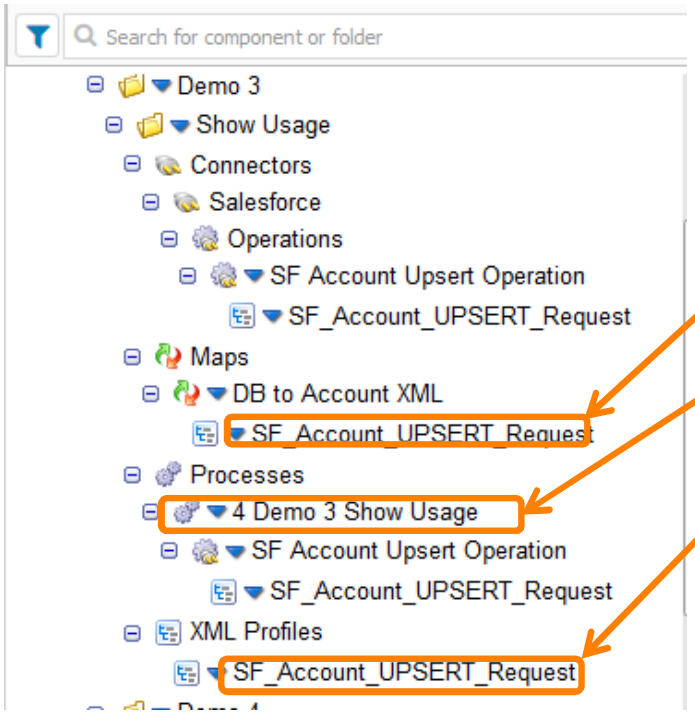
Using “Show Where Used”

- An important part of managing common components is the ability to see where the component is used.
- The Show Where Used feature shows all references of a component using a filter in the component explorer panel



“Show Where Used” Results

The Show Where Used feature shows all references of a component by applying a filter in the Component Explorer panel



- Used in this Map
- Used in this Process
- Reference to the profile itself

Development Life Cycle

- **Folder Management**

Folder Management: Folder Organization

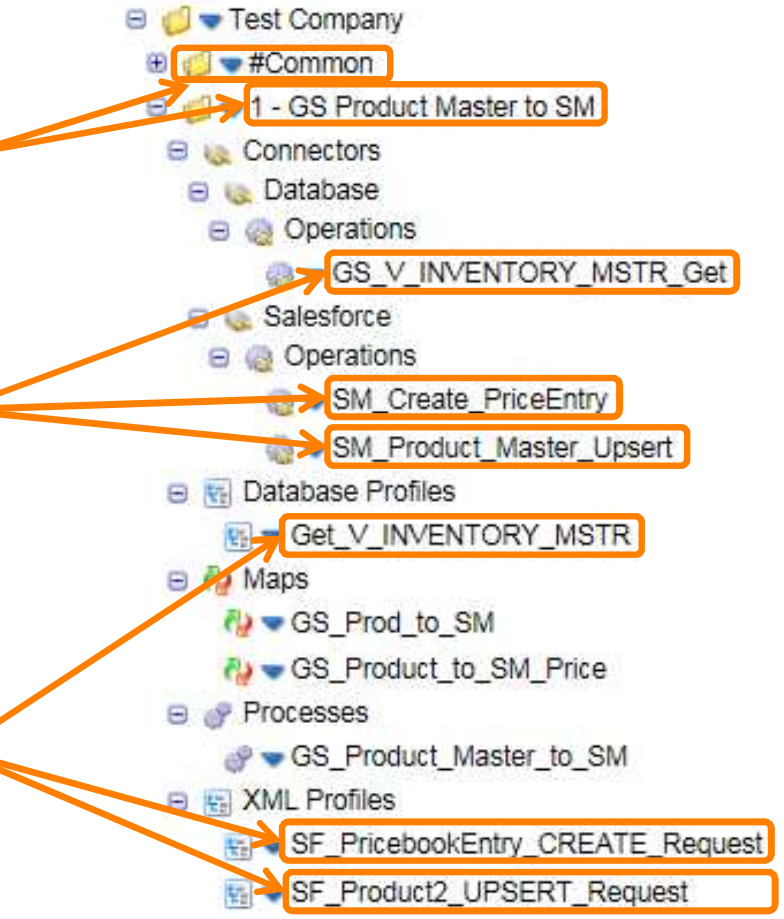
- Determine the folder structure
- Create common components folders
- A three-level structure is:
  - Project, or Phase
  - Integration Group, or Endpoint Group
  - Object, or Object Group
- Projects provide a historical/chronological context for integrations and should use common Operations, Profiles, and Functions.
- The integration group level is optional and may be unnecessary for smaller projects.
- Objects are at a business level and may contain several processes

Folder Organization Example

Use numbers or special characters to force objects to the top alphabetically

Use meaningful names to help identify source and destination objects

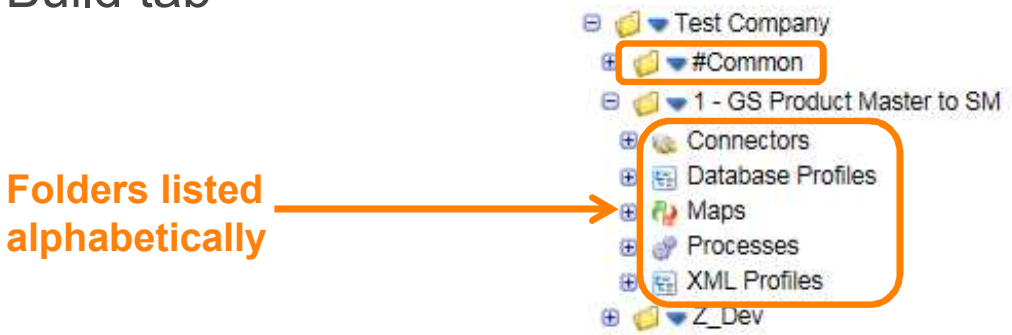
Use meaningful names to help identify actions



Folder Management: Naming Conventions – Special Folders

- This example uses a folder called #Common to contain Connectors:

- In larger projects, a #Common folder could be used in lower level folders, as needed, to contain common Profiles
- Components common at the lowest level reside in the lowest level folder
- Folders are always listed in alphabetical order in the Component Explorer on the left hand side of the Build tab



Folder Management: Naming Conventions – Special Folders

- There are some naming opportunities for handling project “extras”
- Use a best practice of putting a “z_” in front of training, Boomi support, test, and archive folder names, to push them to the bottom
- Use a z_Archive folder to store old Components and Processes that might be useful for reference
- Drag and drop random folders and store them in the z_archive folder. Only folders with live or recent versions should be outside the z_Archive folder.

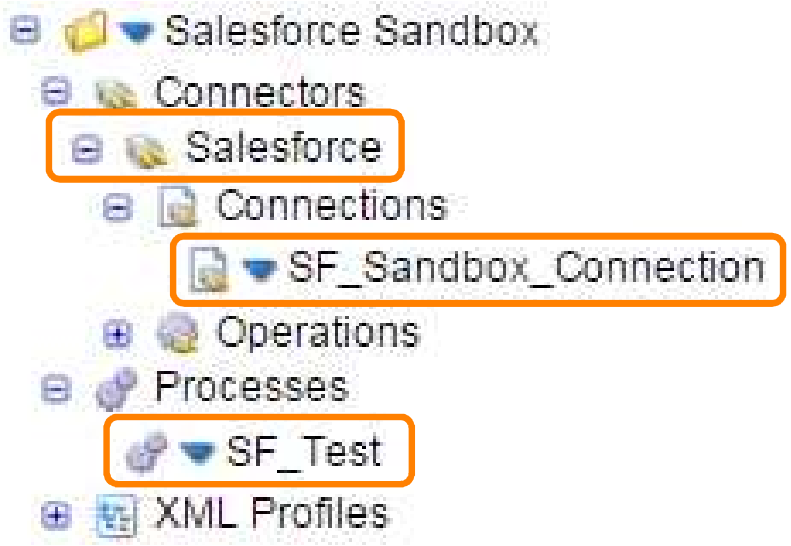
A development folder pushed to the bottom using “_Z”

Folder Management: Naming Conventions – Process Names

- A good Process name should identify:
  - the source system
  - the target system
  - the business object being transported
    - Example: Salesforce Order to NetSuite
- Sub-processes break a large complex process into sub-processes
- Identify the master, or highest level process, as this is the one that is deployed
  - (master) NetSuite Payment Status to Expensewire
  - (sub) Get Concur API token

Folder Management: Naming Conventions – IDs or Endpoints

- Other options include the use of agreed upon component naming conventions.
  - An example might be starting all of the Salesforce related components with an “SF_” to indicate that the components within the folder relate to integrations that start in Salesforce.



Folder Management: Considerations

- Tech tip: view folders as “packages” from an organizational perspective
- Common components that are referenced in multiple places should be “factored out” and moved to a higher-level or common folder

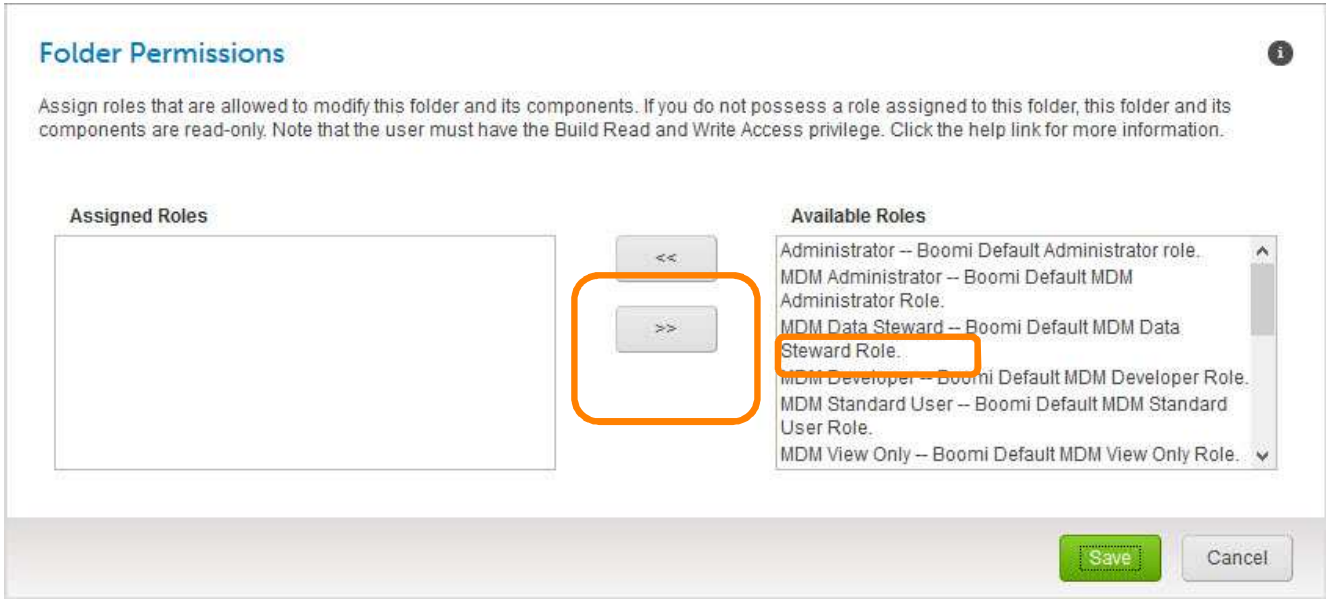
- Organizational considerations:
  - For all projects, create #Common folder under the root folder for Connections
  - Consider low level folders as units of work that you can copy
  - Compile common components at the highest possible factor

Folder Management: Folder Permissions

Used to assign or remove user roles from the selected folder.

Users assigned these roles have write access to the folder and its components.

Users must also have the Build Read and Write Access privilege..



Development Life Cycle

- **Change Management**

How Does Component Versioning Work?

- Each time you save a change to a component, the **Revision** number is incremented along with who changed it
- Each component has its own revision history
- You can roll back to a previous revision – but be careful!
  - Make sure other processes/components are not relying on the newer configuration
  - Rolling back will not delete any other components referenced by the newer configuration
- Show Where Used feature displays all the components that reference a given component

How is Component Source Control Managed?

- AtomSphere is the source control engine; there is no external management or import/export
- There is a single copy of processes and components representing the last revision
- All users work with the latest revision
- Users do not have their own “copy” of the entire code set and contribute changes
- There is no trunk vs. branch concept
- As a graphical configuration tool, there is no automated way to “merge” differences between revisions

What are the Considerations for Development Teams?

- The Build tab is a **shared** development environment
- In Enterprise Editions, use **component locking** feature to prevent concurrent editing (similar to check-out/check-in)
- Even with component locking, communication and coordination across the team is important
- Establish component naming conventions and folder organization
  - Create “sandbox” folders per developer to explore functionality, build proof-of-concepts before incorporating into common development folders
- Each user should install a local Atom for Test Mode and local debugging tools (e.g. proxy)

Environments and Change Management

- An AtomSphere Environment is a dedicated deployment setup that supports different phases of the integration development lifecycle
- **Environments** are logical deployment "containers" intended to model traditional life cycle stages:
  - "Dev" – End-to-end testing by Development team
  - "Test" – End-to-end testing by QA/users
  - "Production" – Live production processes
- They enable change management and allow you to run different versions of the same process simultaneously and avoid making copies of processes and components
  - Example: Version 1 running in production while version 2 is being tested

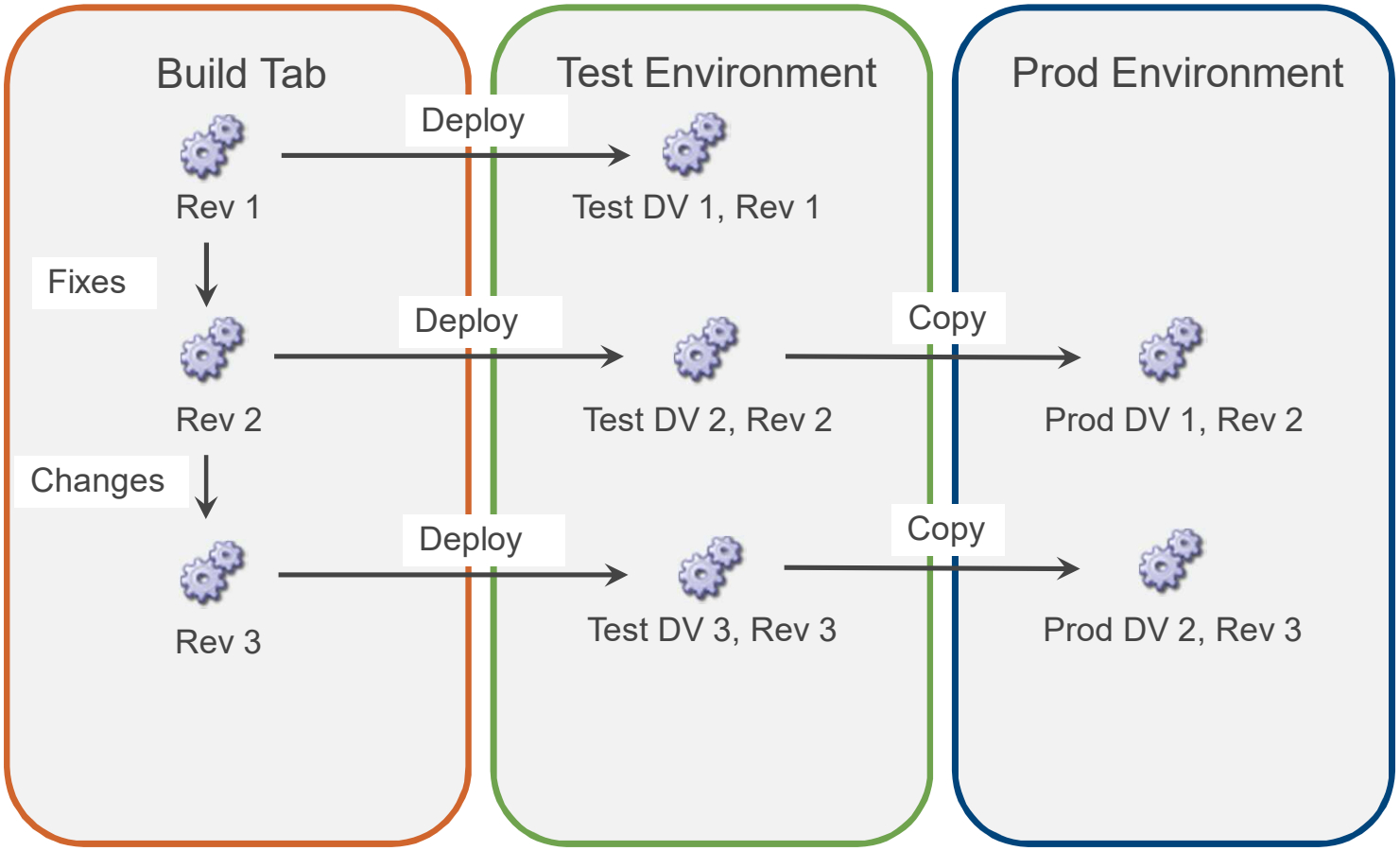
Process Deployment and Change Management

- Deploying a process creates a **Deployment Version** which is a “snapshot” of the process and its components
- Processes are deployed to a specific Environment
- The deployed process version is isolated from subsequent changes to components in the Build tab
- Must redeploy the process to publish new changes to Environment
- Some processes must be deployed to be scheduled or to listen for real-time requests
- Processes can be deployed individually or in bulk
- Deployment Versions can be rolled back in the event of a “production bug”

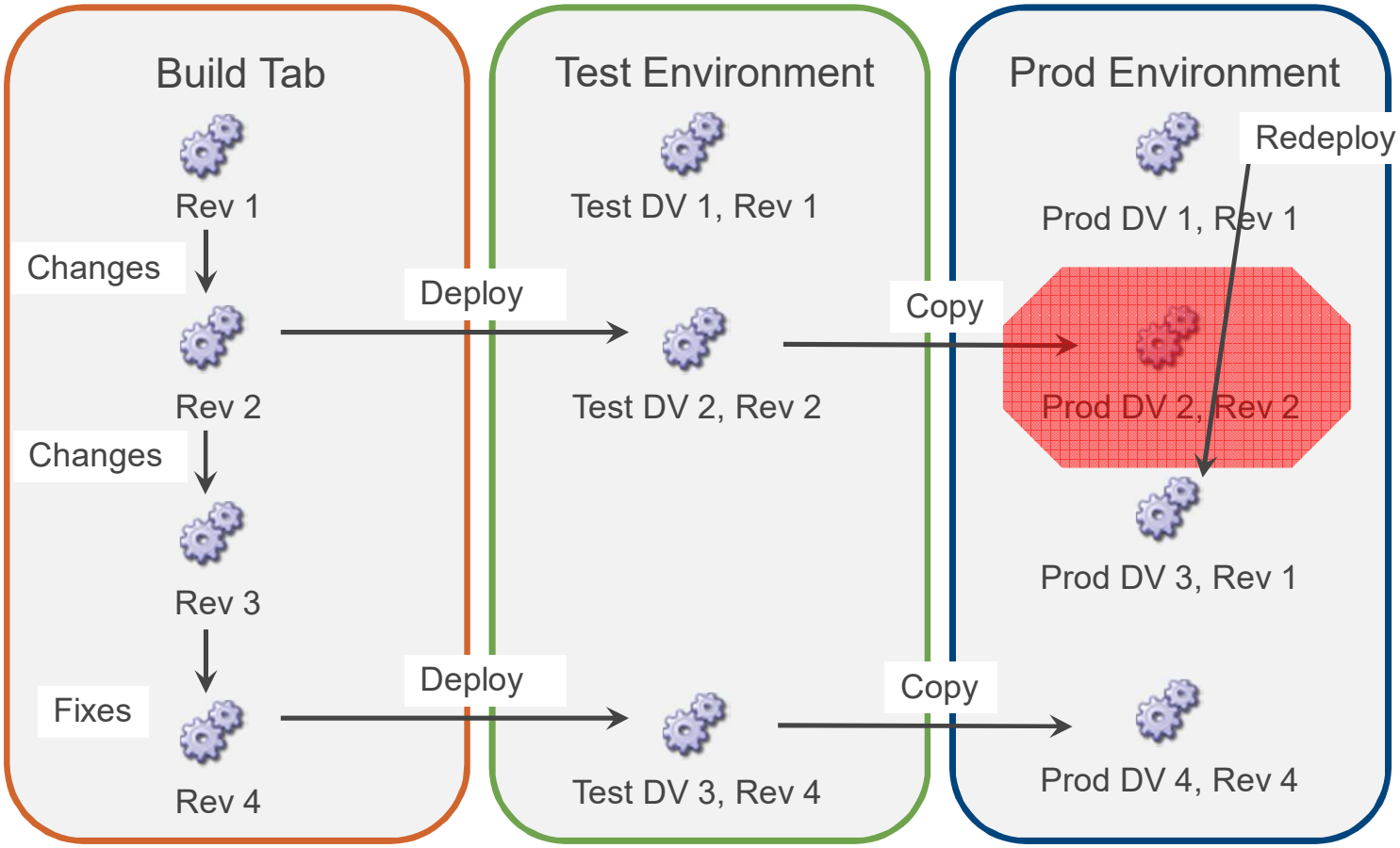
Component Dependencies, Deployment, and Change Management

- Only the process component itself is deployed
- When deployed, the process and all components referenced by the process or other components are “bundled” together
- Processes are deployed independently
- Observations:
  - Common components
    - Suppose Process 1 and Process 2 both reference Map A and are deployed. If you change Map A but only re-deploy Process 1, Process 1 will be using Map A revision 2, while Process 2 will still be using Map A version 1.
  - Child/Sub processes
    - Child processes are included in the deployment of the parent process
    - If a child process is to be deployed (e.g. to be able to retry docs), or if you make a change to a child process you should always deploy both parent and child to avoid version mismatch

What does change management look like in action (normal)?



What does change management look like in action (rollback)?




Properties
Boomi

What are Properties?

Properties consist of name/value pairs or variables used to store specific information to assist with the integration.

Two Types:

- Document-level
- Process-level

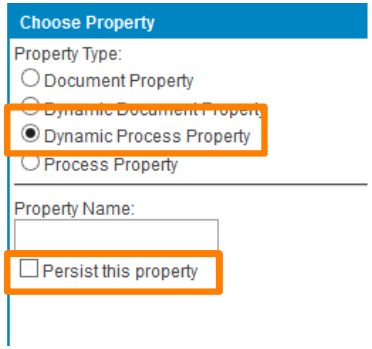
Document-Level	Process-Level
• Document Properties ◦ Connector and Trading Partner Information ▪ Inbound (read-only), outbound (write-only) ◦ Meta Information ▪ Set by certain shapes ▪ Read-only • Dynamic Document Properties ◦ Arbitrary name/value pairs ◦ Read/write	• Process Property component ◦ Pre-configured component ◦ Defined data types ◦ Read/write • Dynamic Process Property ◦ Arbitrary name/value pairs ◦ Name can be dynamically-specified ◦ Read/write

Properties

- **Dynamic Process Properties**

Dynamic Process Properties

- Set a property value in the beginning of the process and retrieve it later in a different part of the process.
- Available across other processes initiated via the Process Call shape, as is common in parent/child process designs.
  - Property set in Parent is available to Child
  - Property set in Child is available to Parent (after execution)
- A process property value can be “persisted” and the value "remembered" for future process executions (using the same process/Atom).

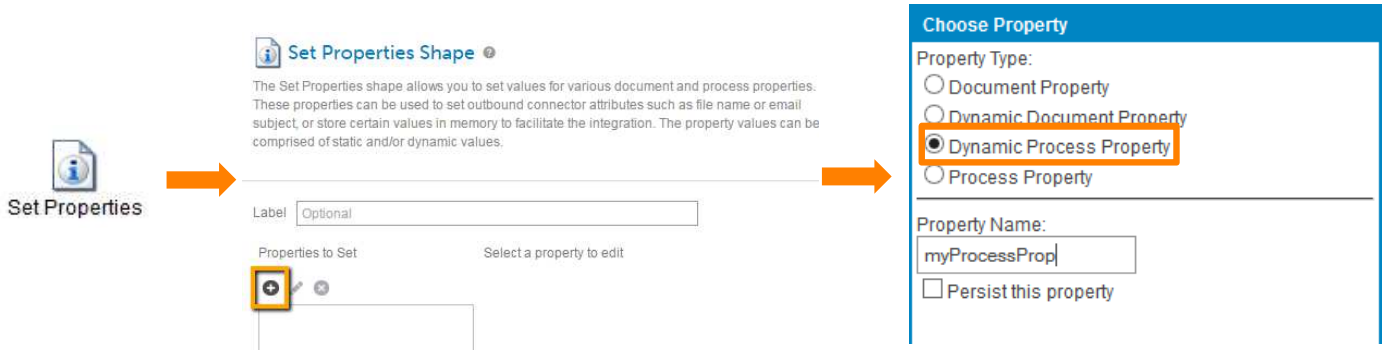


Dynamic Process Properties

Can be set in:

Set Properties shape

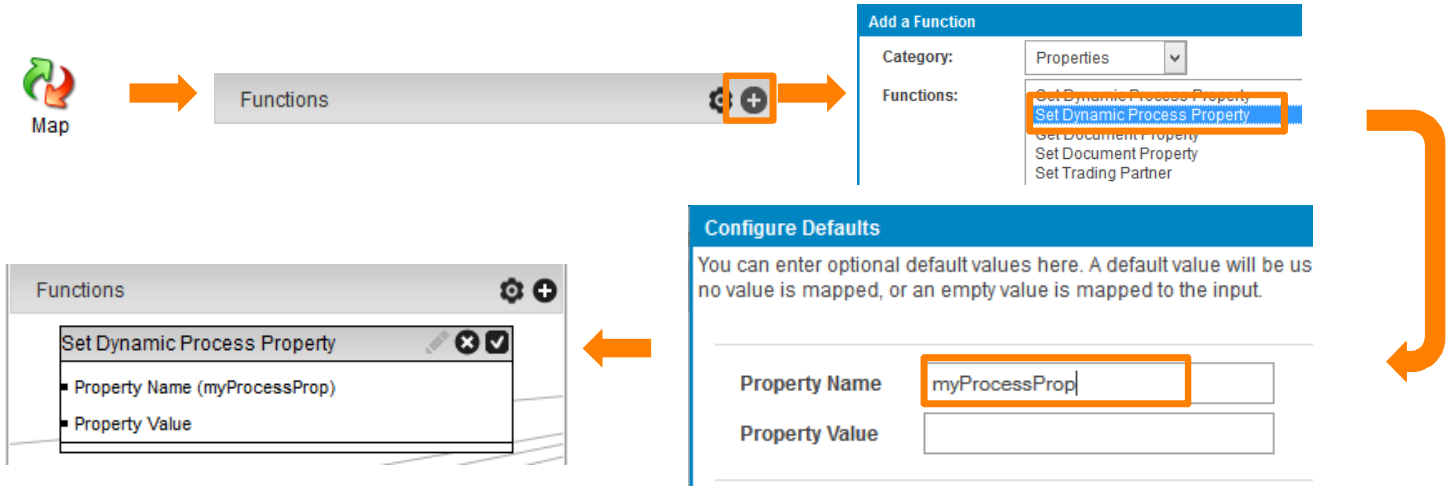
Make special note of the assigned Property Name (spelling, case sensitivity, etc.). You will need to reference it later when attempting to retrieve it.



Dynamic Process Properties

Can be set in:

- Set Properties shape
- Set Dynamic Process Property map function



Dynamic Process Properties

Can be set in:

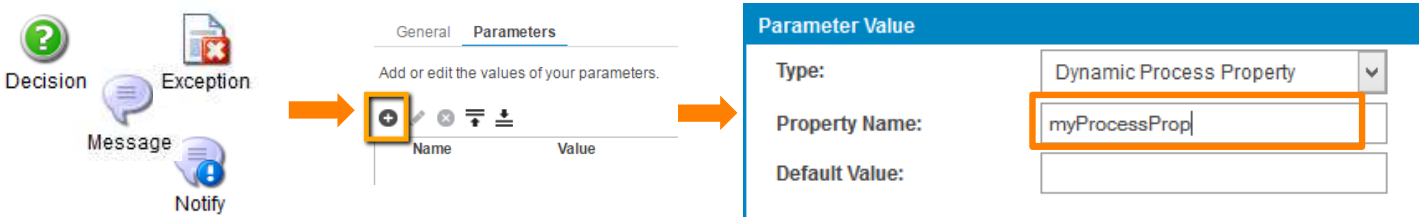
- Set Properties shape
- Set Dynamic Process Property map function
- Groovy script


```
// Import the ExecutionUtil class
import com.boomi.execution.ExecutionUtil;

// Retrieve a Process Property value
String myPropValue = ExecutionUtil.getExecutionProperty("MY PROPERTY NAME");

// Set a Process Property value
ExecutionUtil.setExecutionProperty("MY PROPERTY NAME", myPropValue);
```

Dynamic Process Properties

Can be retrieved from:

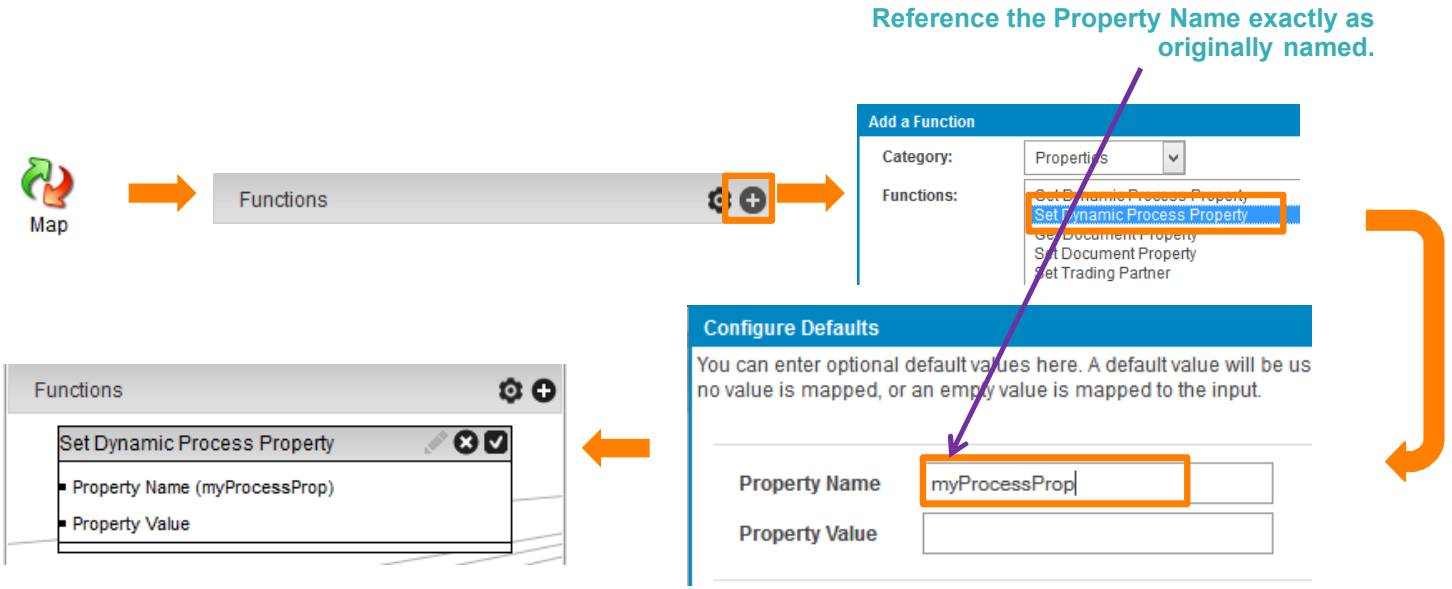
Parameter Type list from most process shapes



Dynamic Process Properties

Can be retrieved from:

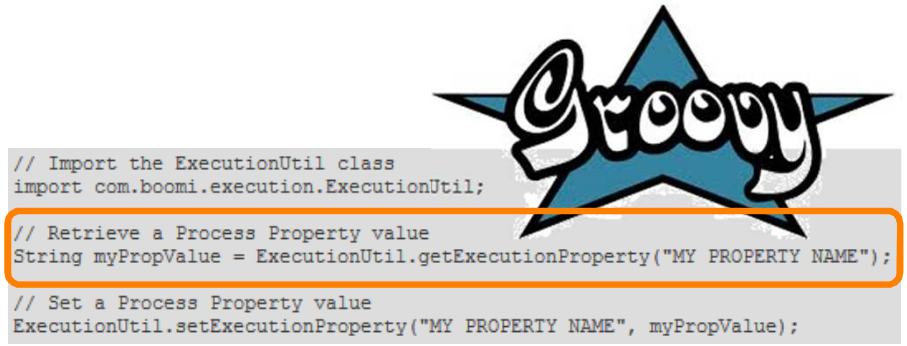
- Parameter Type list from most process shapes
- Get Dynamic Process Property map function



Dynamic Process Properties

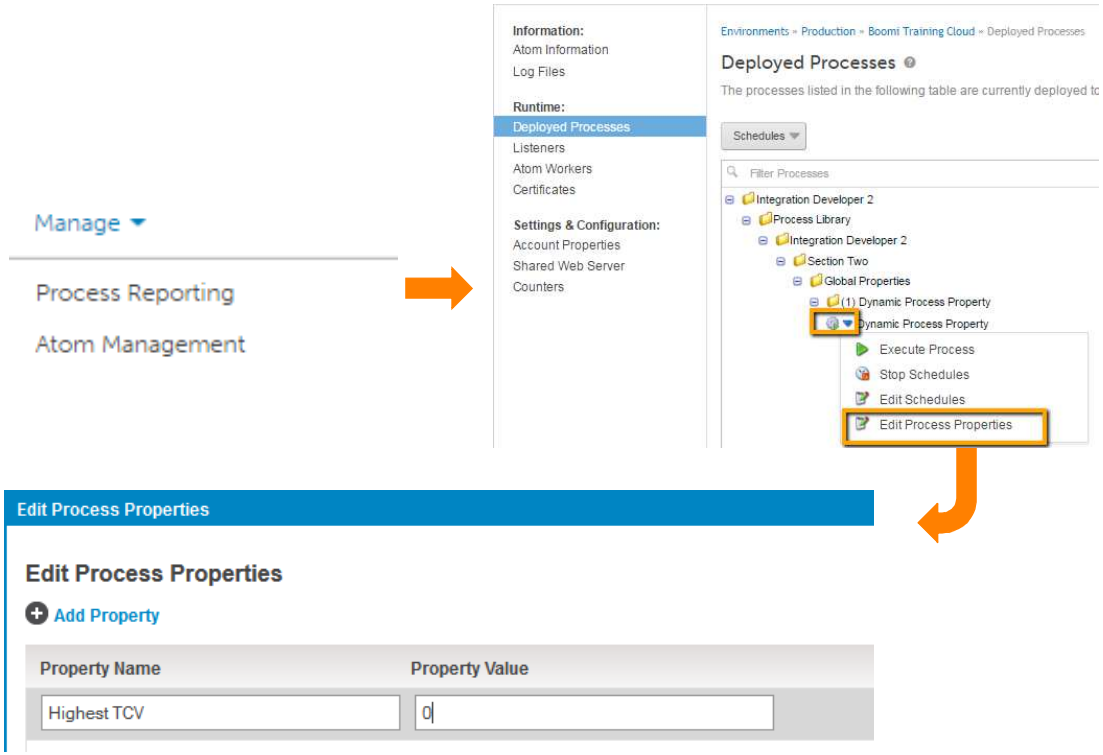
Can be retrieved from:

- Parameter Type list from most process shapes
- Get Dynamic Process Property map function
- Groovy script



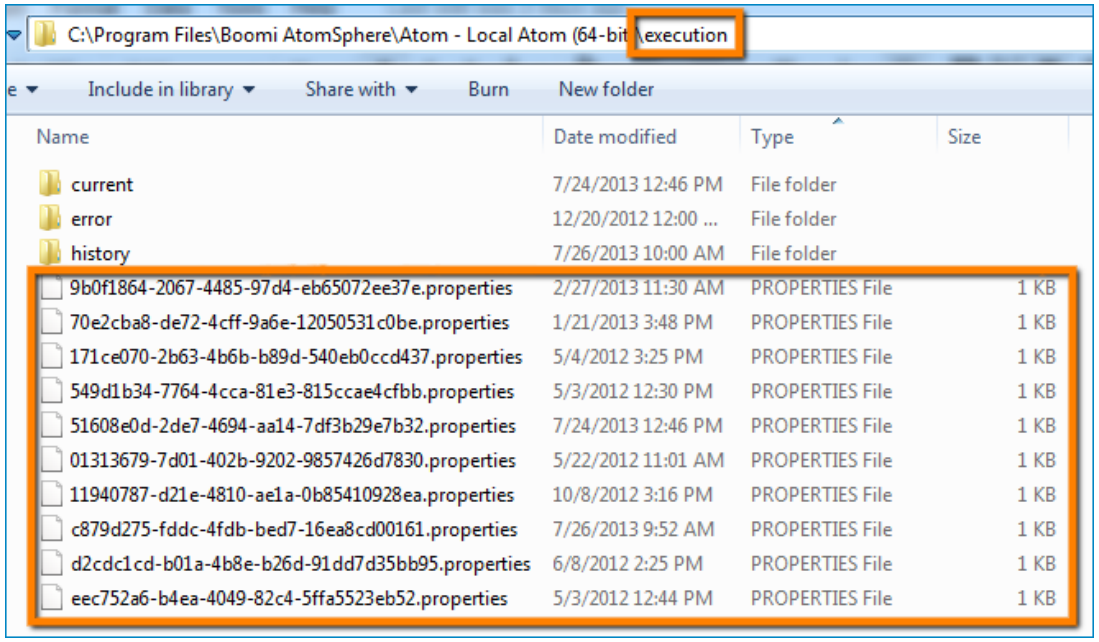
Dynamic Process Properties

Persisted Process Properties of Deployed Processes can be edited or deleted within: **Manage > Atom Management**



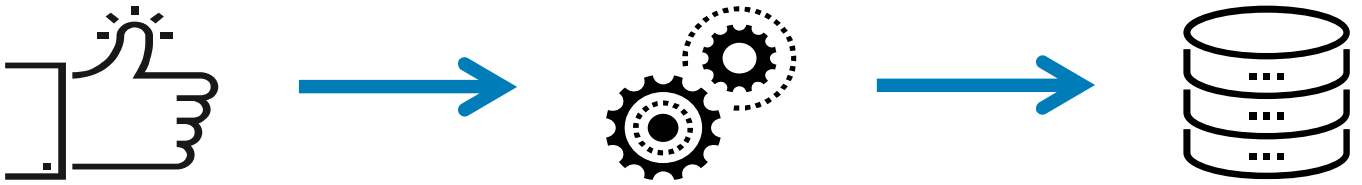
Dynamic Process Properties

Persisted Process Properties of Deployed Processes are stored in the **execution** directory of the attached Atom.



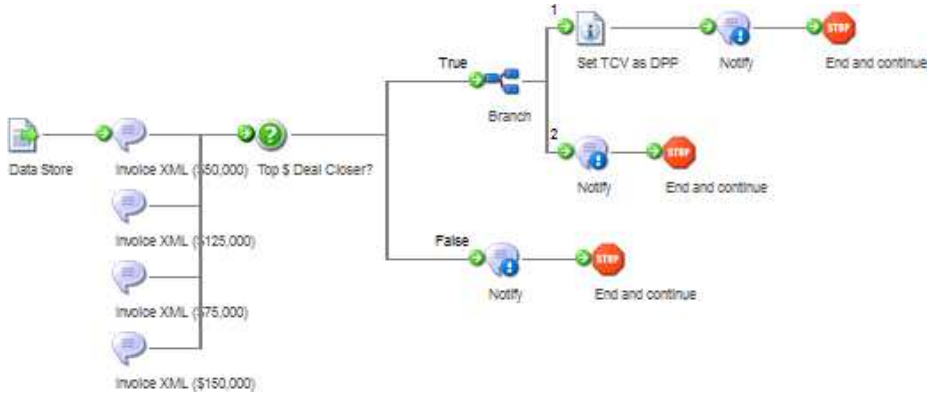
Business Use Case for Activity

- Each week, a company awards the sales rep who closes the largest deal that week
- A process continually checks for newly created invoice records in the company's data store to track the highest producing deal
- The data store contains processed invoices containing basic account information, including the sales rep and the closed-won opportunity's Total Contract Value (TCV)



Dynamic Process Property Activity

- Process individual invoices containing account information
- Compare invoice's Total Contract Value (TCV) to highest TCV stored in memory (add a Dynamic Process Property)
- Store new TCV in memory if it is greater than highest TCV currently stored as Dynamic Process Property (DPP)
- Configure notification details using a Notify Shape in preparation for advanced logging and reporting



Participants to Complete

Participants to Complete:

Dynamic Process Properties Activity

Page(s): 35-44

Instructor to Review

Instructor To Review:

Dynamic Process Properties Activity

Page(s): 35-44

Properties

- **Dynamic Document Properties**

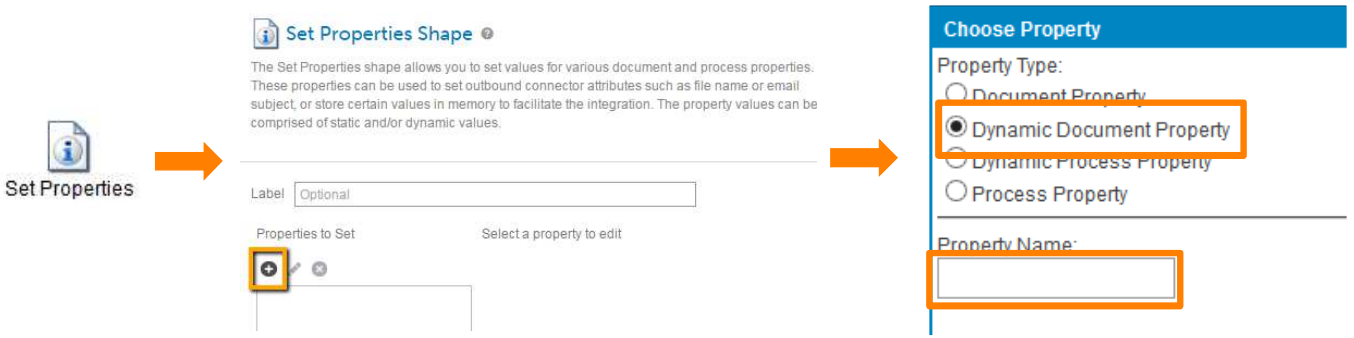
Dynamic Document Properties

- Define and temporarily store additional pieces of information about a given document.
- Specific name/value pairs follow the document through its execution.
- Unlike a process property, you cannot persist a document property value and “remember” its value for future process executions.

Dynamic Document Properties

Can be set in:

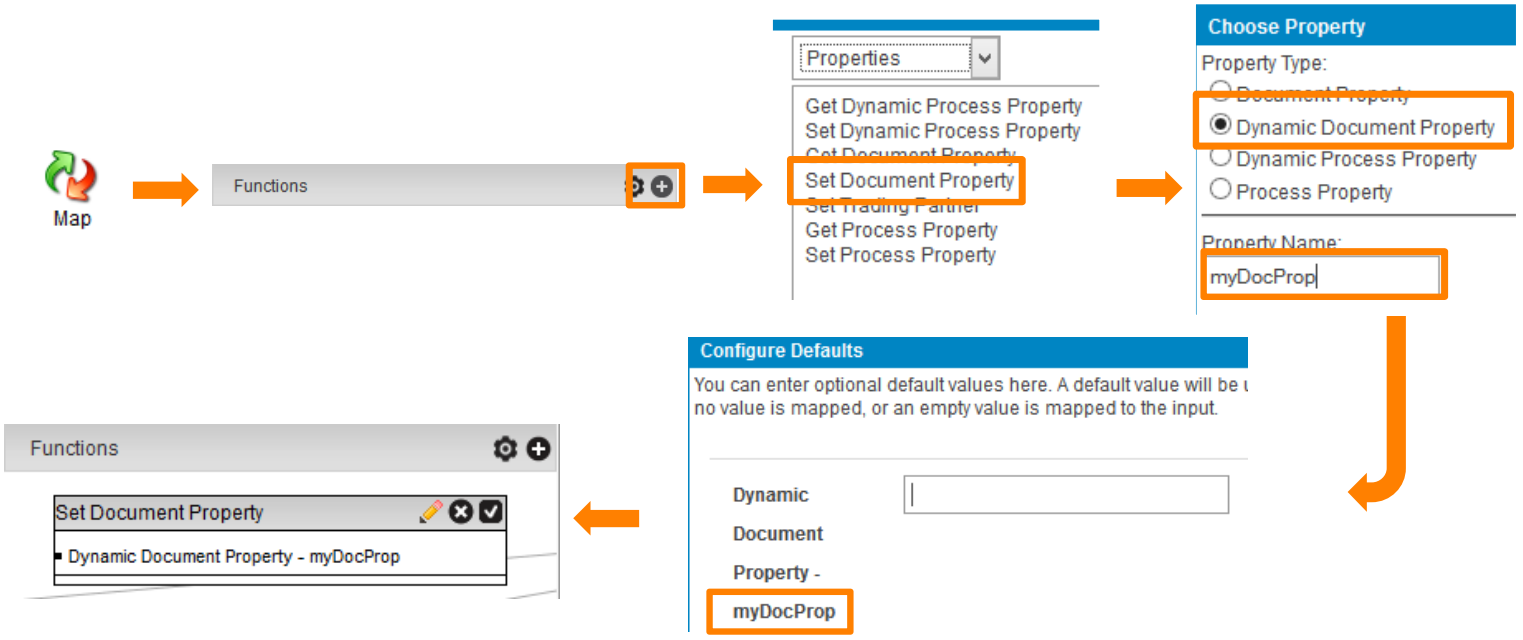
Set Properties shape



Dynamic Document Properties

Can be set in:

- Set Properties shape
- Set Document Property map function



Dynamic Document Properties

Can be set in:

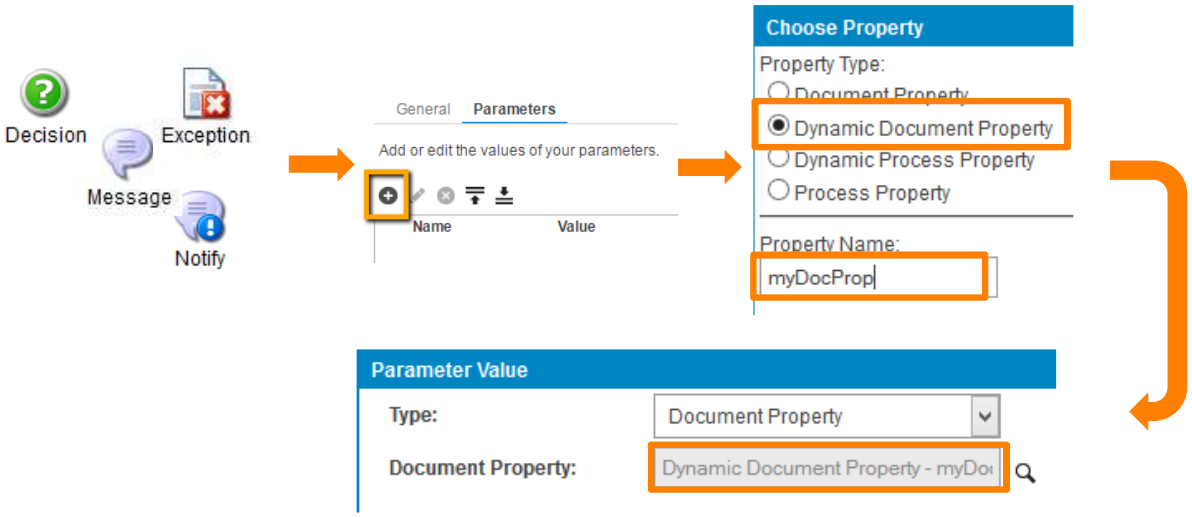
- Set Properties shape
- Set Document Property map function
- Groovy script


```
// Store the new property value
props.setProperty("document.dynamic.userdefined.MY_PROPERTY_NAME", propValue);
dataContext.storeStream(is, props);
```

Dynamic Document Properties

Can be retrieved from:

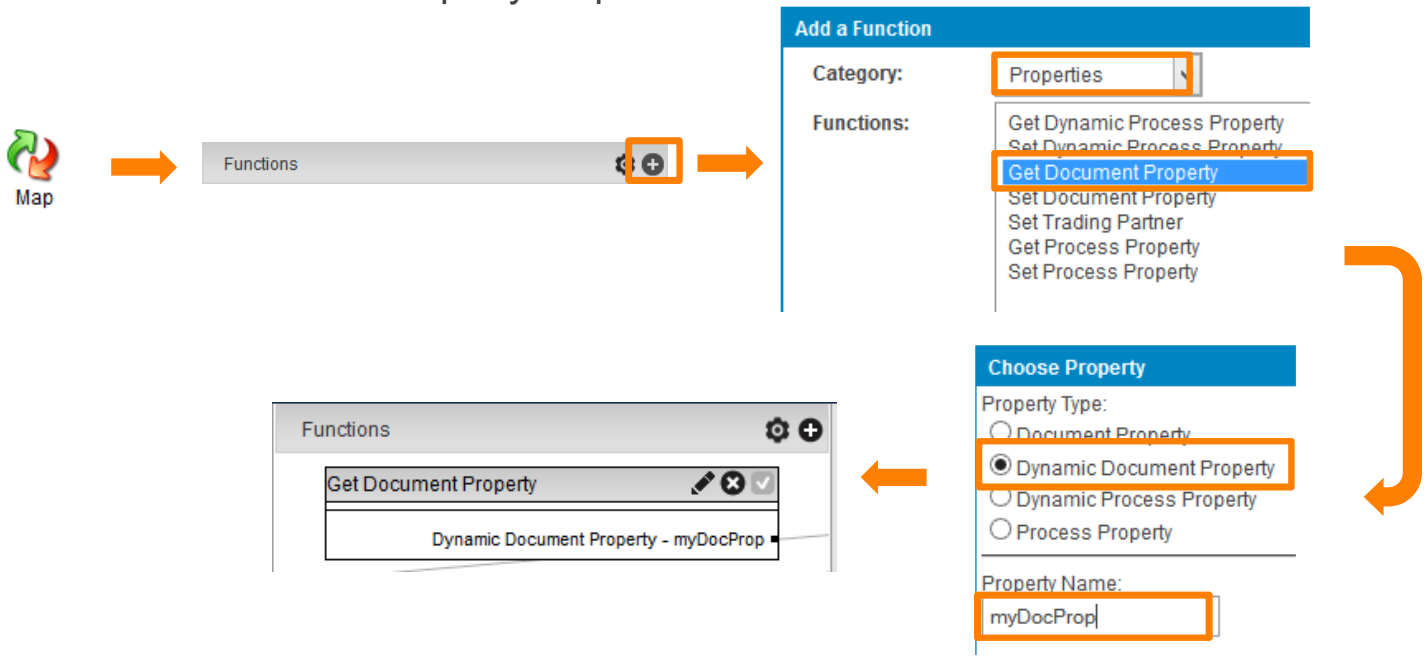
Parameter Type list from most process shapes



Dynamic Document Properties

Can be retrieved from:

- Parameter Type list from most process shapes
- Get Document Property map function



Dynamic Document Properties

Can be retrieved from:

- Parameter Type list from most process shapes
- Get Document Property map function
- Groovy script

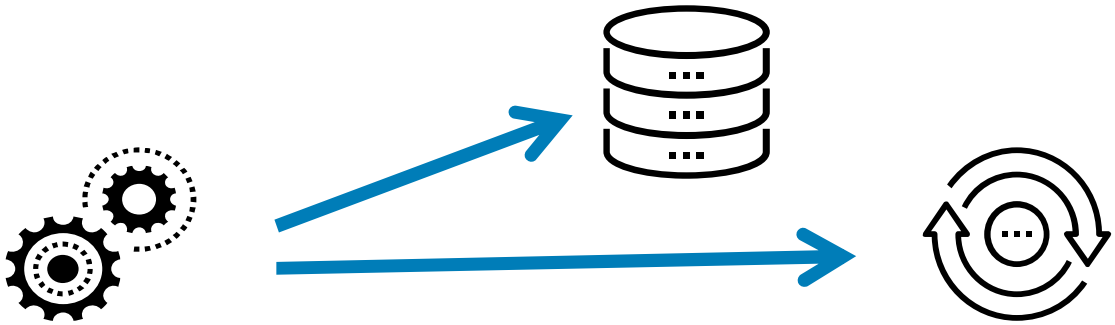

```
// Retrieve the current property value
propValue = props.getProperty("document.dynamic.userdefined.MY_PROPERTY_NAME");
```

Differences Between…

Dynamic Process Properties	Dynamic Document Properties
Value can persist across process executions	Value cannot persist
Value is assigned on the process level – the value remains the same unless changed by a process step	Value is assigned on the document level and can be different per document
Once set, available anywhere in the process including child processes	Once set, only available as long as the document exists – will continue across branches (if set before the Branch shape), but does not continue across Message shapes or outbound Connectors

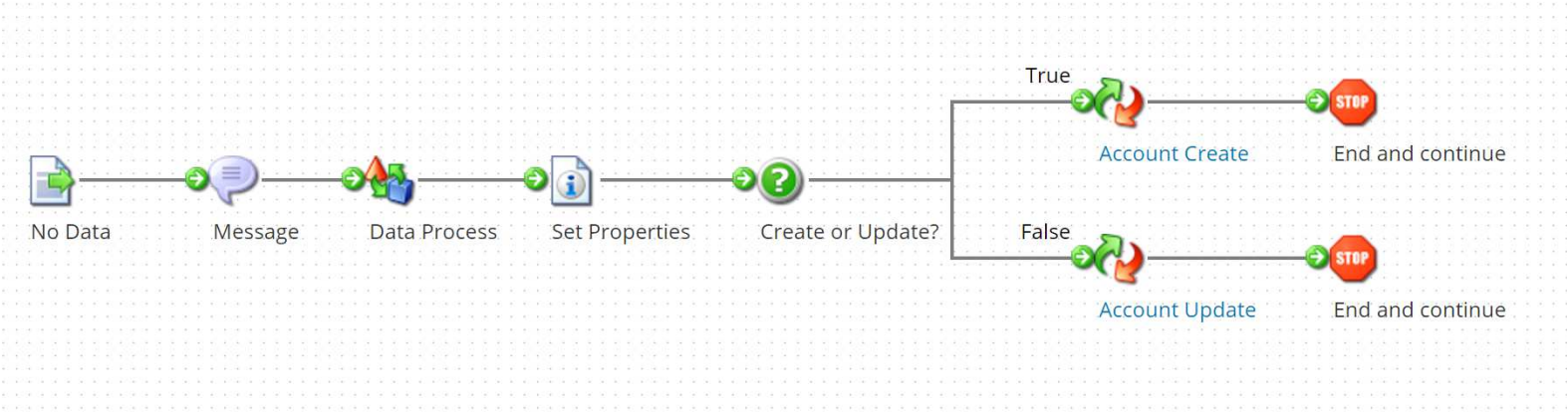
Business Use Case for Activity

- A process continually checks if a given Account record exists in a company's data storage system to determine whether to perform a Create or Update call
- The process retrieves an Account's internal system ID using a Connector Call output parameter as a Dynamic Document Property and then creates a new Account or updates an existing Account



Dynamic Document Property Activity

- Add a Connector Call in the Set Properties shape
- Configure the Decision shape
- Configure the Create and Update Maps
- Run a Test and observe the results



Participants to Complete

Participants to Complete:

Dynamic Document Properties Activity

Page(s): 45-51

Instructor to Review

Instructor To Review:

Dynamic Document Properties Activity

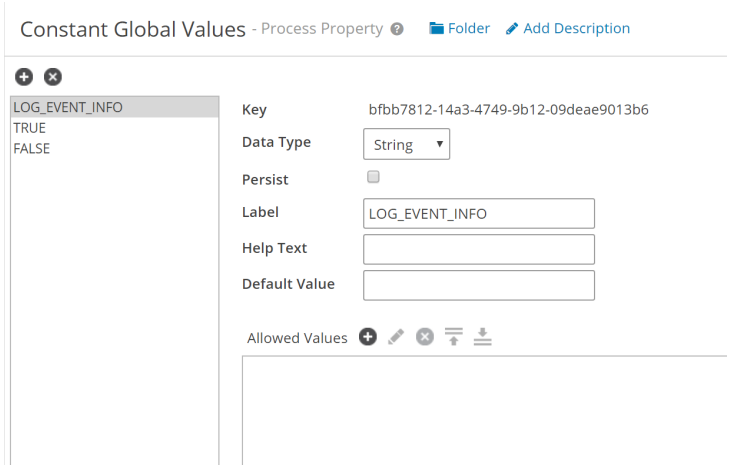
Page(s): 45-51

Properties

- Process Property Component

Process Property Component

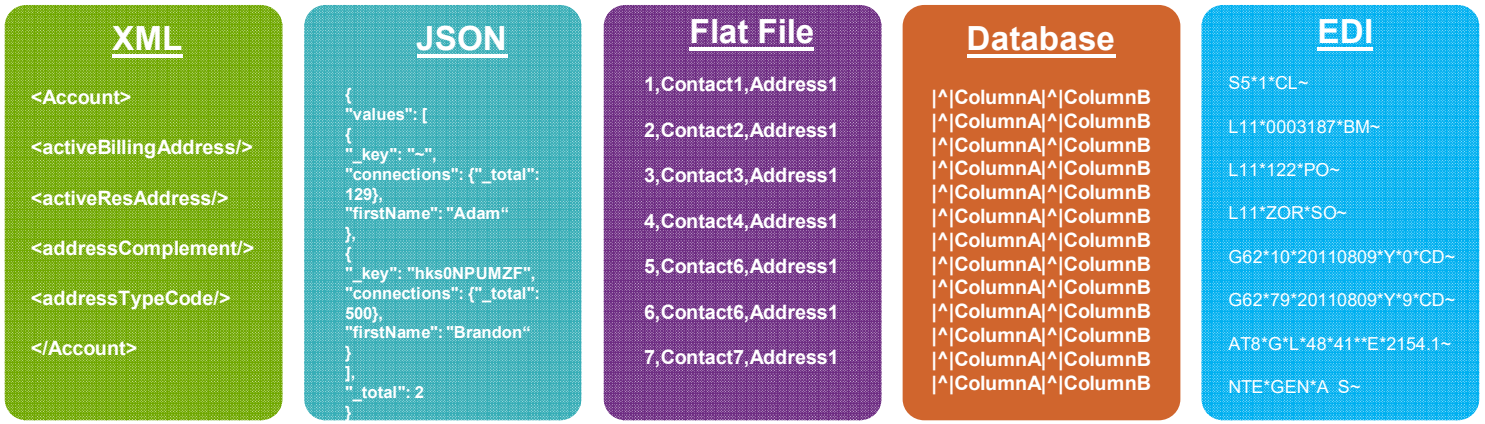
- Group the various settings, default values, and/or persisted properties for the given process
- Convenient to use with environment extensions
- Define static, constant values to reference consistently throughout your processes
- A Process Property is a component, therefore it has its own revision history




Document Flow
Boomi

Review: What is a Boomi Document?

- It is the main unit that powers the execution in a process flow.
- Dell Boomi supports five raw document types:

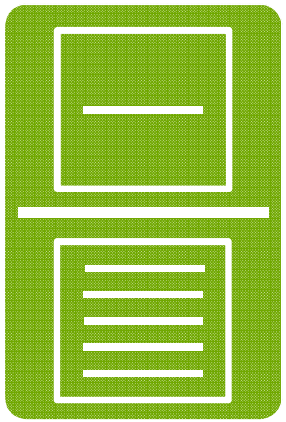


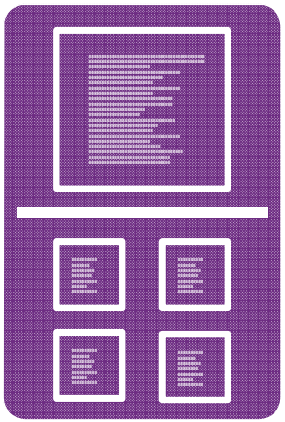
- A document is not necessarily a logical or physical record
- Document definition depends on the data source and profile

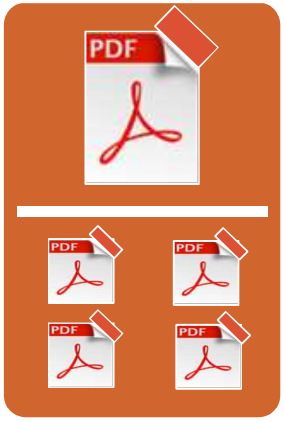
What is a Document?

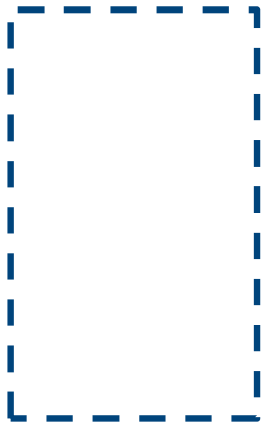
Documents are presented in four different formats:

- Records
- Transactions
- File Instances
- Empty









Document Flow: Concepts

- Shape Execution
- Execution Path
- Documents with Multiple Records
- Original Document ID
- Document Failures
- Altering Document Flow

Shape Execution

Shape Execution refers to how documents move through the process shapes together

Behavior

- Documents move as a group from shape to shape. Each document executes on a given shape **IN SEQUENCE** and generally **INDEPENDENTLY** before any of them move to the next shape.
- A shape executes only if at least one document reaches it.
  - For example: if a start shape does not return any documents (e.g., nothing matches query), no subsequent shapes are executed.
  - Note that a No Data start shape actually generates a single, empty document.
- Documents never affect anything that happens in a prior execution shape, as all documents have already passed that shape.
- Processing as a group is more efficient because the shape logic is loaded into memory once for the entire group of documents.

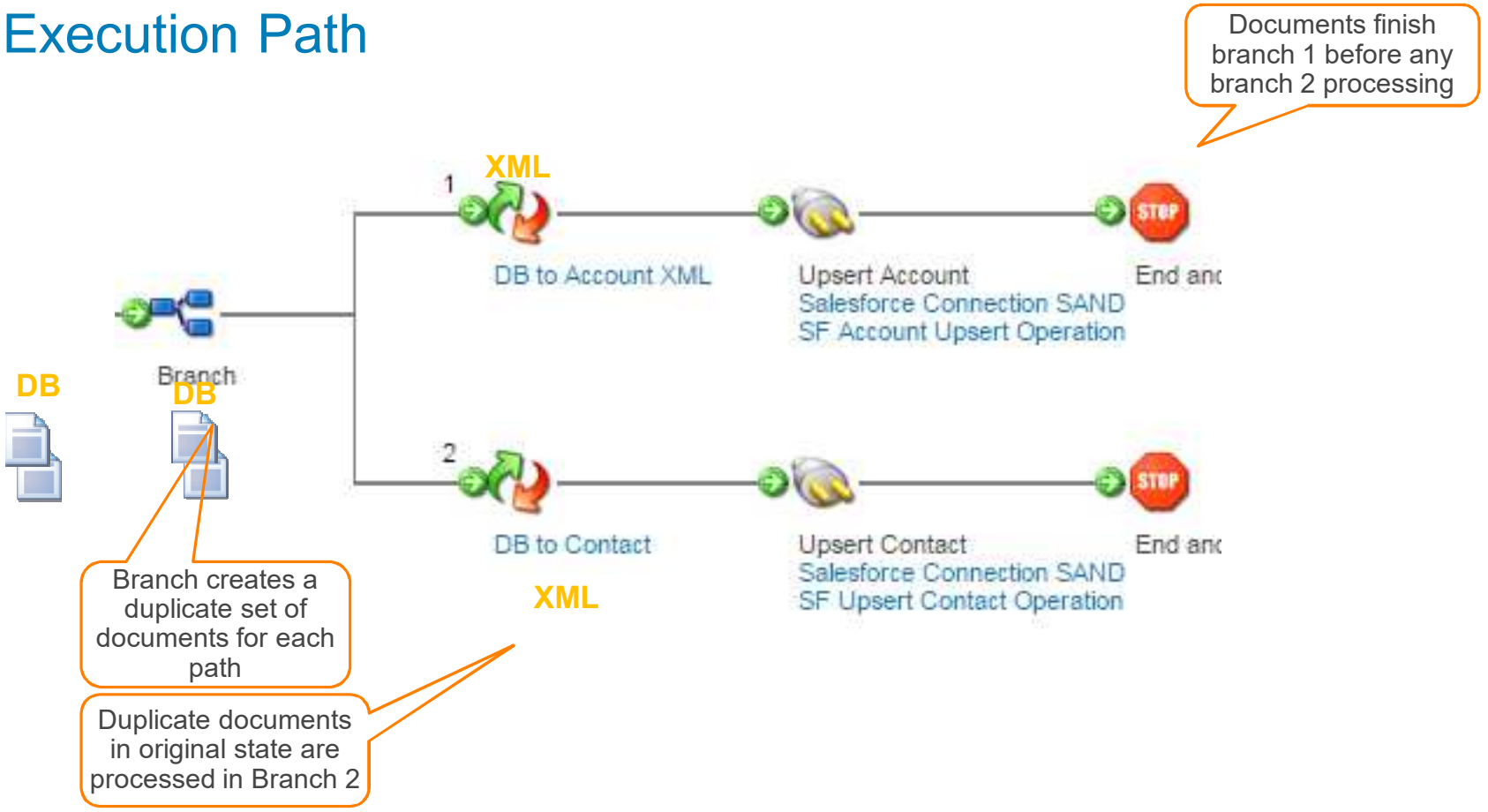
Execution Path

Execution Path refers to the route documents take through the process

Behavior

- Documents move from the start shape to one or many end points.
- All documents do not have to take the same path: Decision, Route, and Business Rule shapes cause documents to take different paths.
- Paths execute sequentially, not in parallel. A path completes before the next path starts.
- Path independence:
  - Branches cannot communicate document changes (data or properties).
  - Must use other techniques to pass data to next path.

Branch Execution Path



Documents Containing Multiple Records

A single document may contain multiple logical records

Implications

- Shapes configured to reference individual profile elements (e.g., Decision) will use the value from the first record in the document.
- Shapes that reference the entire profile (e.g., Map) will iterate through all logical records.

Use Cases

- Any type of data can be a “batch” document, most common with flat file (CSV) and database data.
- Records within a single document can be aggregated in Maps.

Original Document ID

Each document read into a process via the start shape is assigned an internal/unpublished identifier, known as the original document ID. This identifier follows the document through the process. Why is this important? Primarily to understand how document failures behave and to track documents.

Key points

- Splitting a document propagates the original ID to the resulting documents.
- Combining documents consolidates the original IDs.
- Connector calls within the process create a new set of document IDs. (However, there is still traceability from a document failure perspective).
- In deployed process logs, the “view linked documents” command can be used to associate different sets of documents with common IDs.

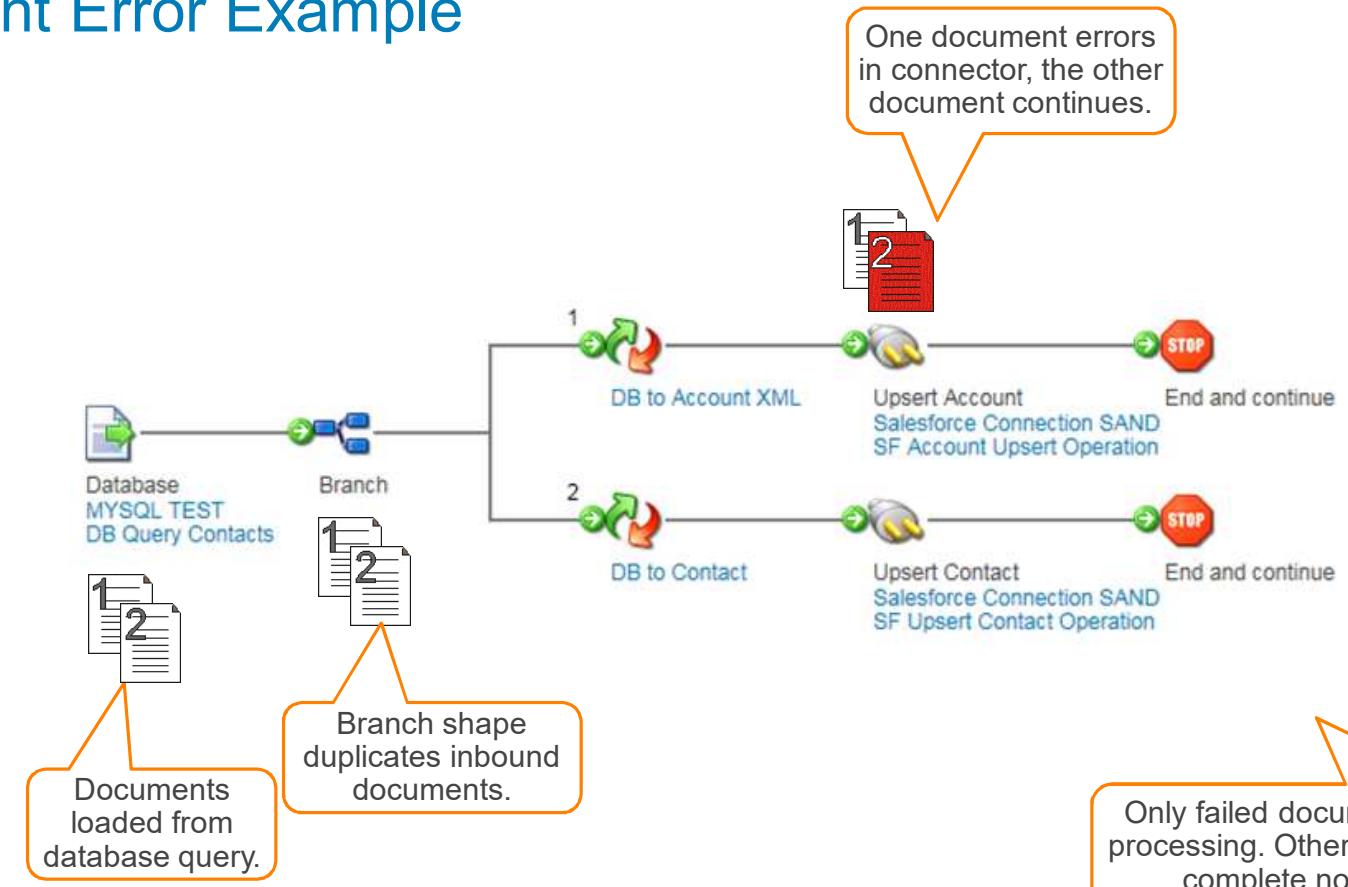
Document Failure

When an exception occurs, the affected document(s) stops immediately and does not continue to subsequent shapes.

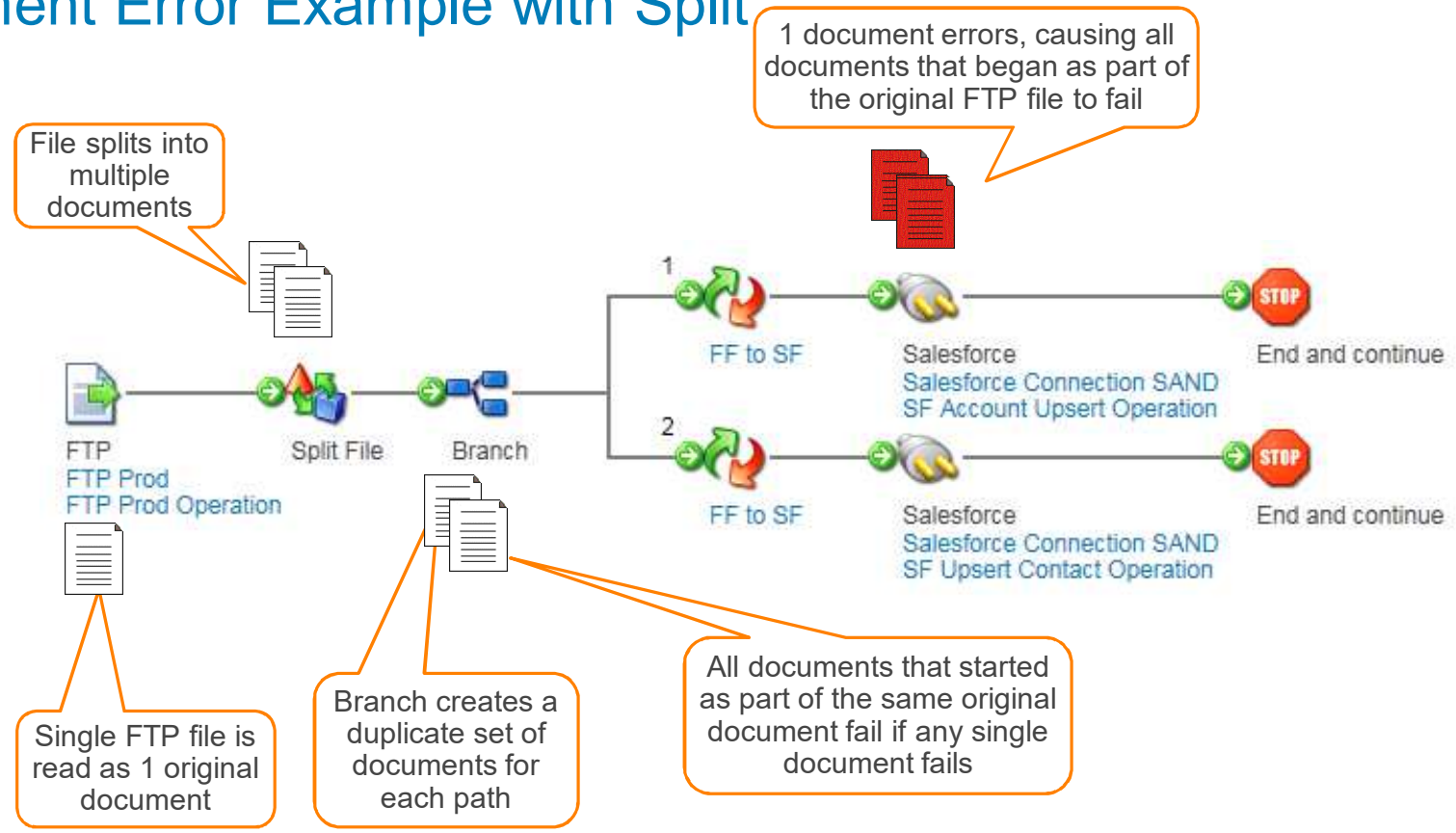
Behavior

- Exceptions can occur per-document or for the entire process.
  - "Process-level" exceptions stop all documents immediately.
  - "Document-level" exceptions stop the individual document(s) but other documents continue processing.
- Documents are identified by their original ID. This is especially important to remember when splitting a document with multiple records or when combining documents:
  - Split: If one of the split documents encounters an exception, the single original document is marked as failed and stops processing. *This means all the other split documents from that original document will stop processing as well.*
  - Combine: If a combined document encounters an error, all the original documents are marked as failed and stop processing.

Document Error Example



Document Error Example with Split

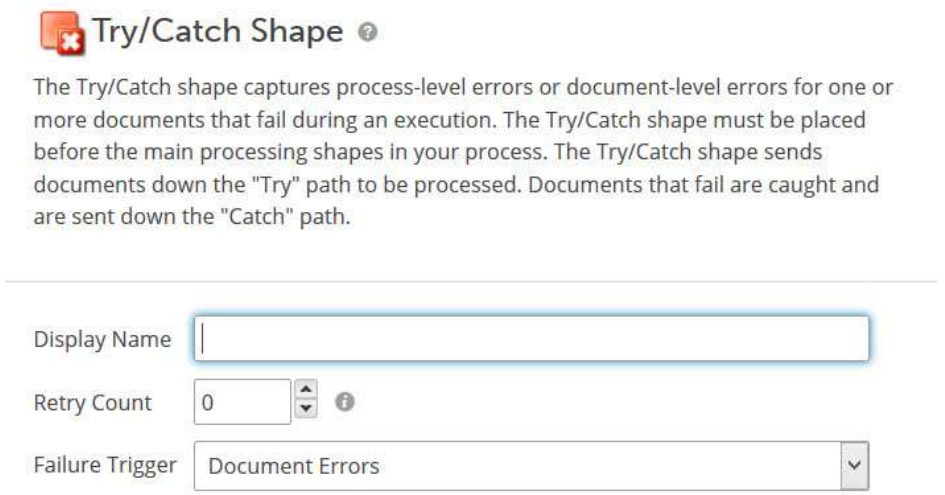


Concepts: Altering Document Flow

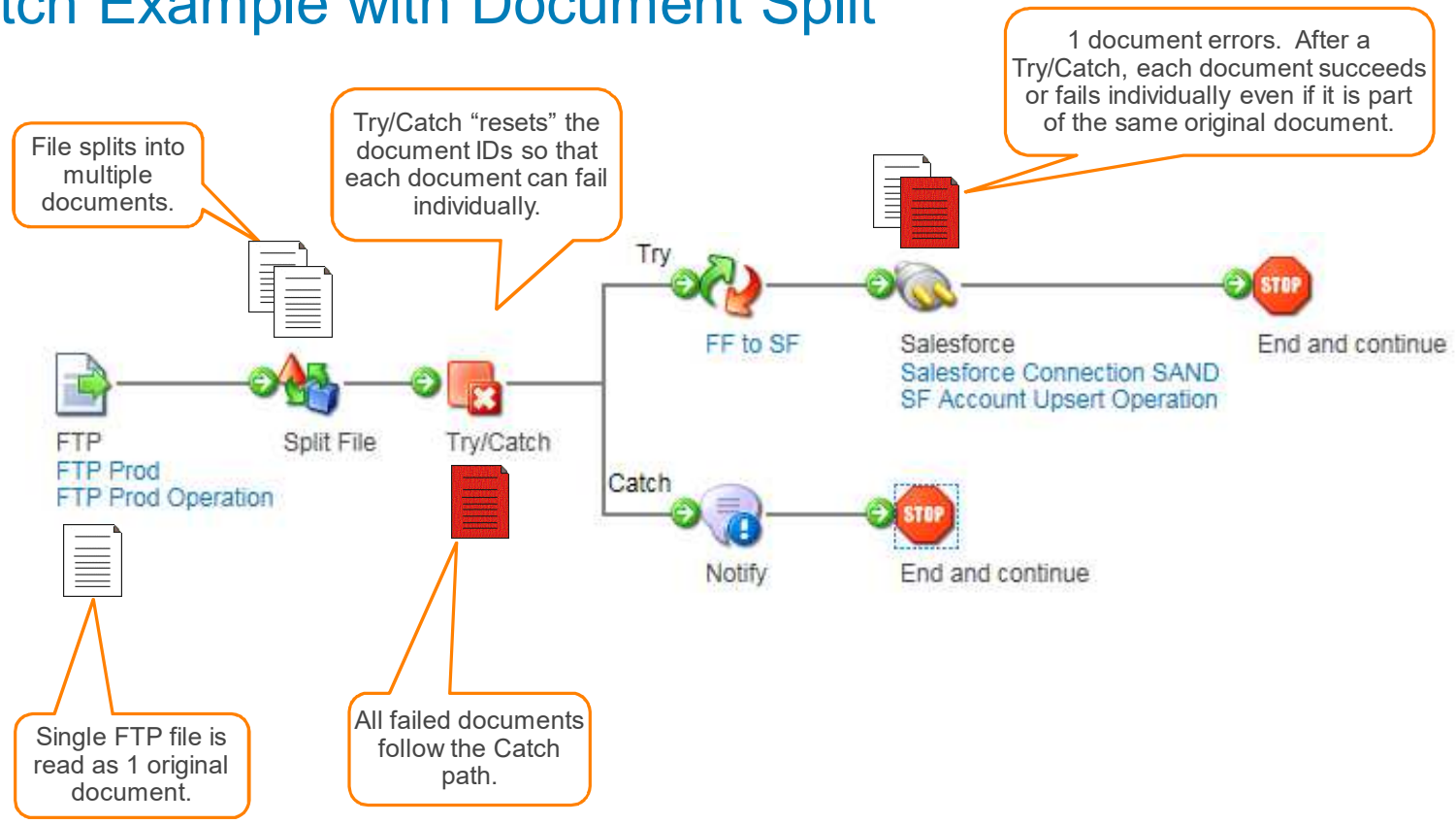
The Try/Catch shape can alter how documents with multiple records flow through the process.

Configuration and behavior:

- The Try/Catch automatically resets a document's Original Document ID so that it can fail individually.
- All documents are sent down the Try path.
- Only documents that fail down the Try path will then flow down the Catch path, even if they came from a split batched document.



Try/Catch Example with Document Split



Altering Document Flow

The Flow Control shape can alter how documents flow through the process

Configuration and behavior

- Execute documents individually or as an arbitrary group size vs. as a single group.
- Execute documents simultaneously in multiple threads vs. a single thread.
- Each document executes to completion before next document starts, one at a time, subset at a time, and/or across multiple threads.

Implications

- Notable performance decrease as each shape's logic is initialized multiple times vs. once per group so only use if appropriate for scenario.

Use Cases

- Cases where one record post impacts the next.
- Incremental inventory counts.
- Detecting duplicates across documents with the group.

Participants to Complete

Participants to Complete:

Document Flow Activity

Page(s): 52-65

Instructor to Review

Instructor To Review:

Document Flow Activity

Page(s): 52-65

Document Flow: Shape Types

- Simple shapes
- Conditional shapes
- Branch shapes
- Connector shapes
- Data modification shapes
- Data Process shapes
- Process Call and Return Documents shapes
- Path completion shapes

Simple Shapes

Simple shapes perform actions that do not change the document contents or flow

Shape types

- Set Properties, Notify, Program Command

Behavior

- Do not modify document flow or document contents.
- Executed once per document that reaches it.
- Note: Notify has a “Write Once per Execution” option, meaning it will execute once per document group vs. per individual document.

Conditional Shapes

Conditional shapes perform logic to assign documents to one of several result paths

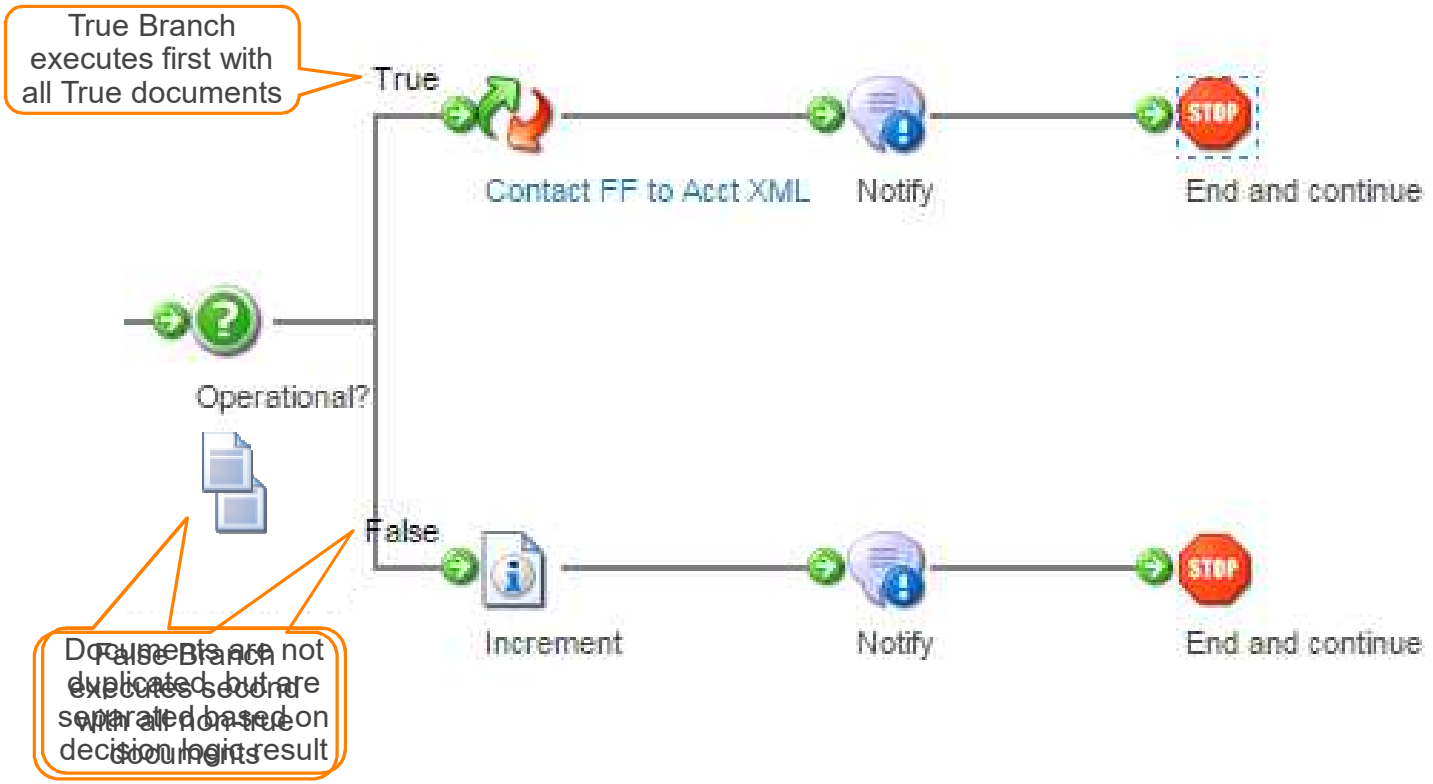
Shape types

- Decision, Route, Business Rules, Try/Catch, Cleanse, Find Changes

Behavior

- Each document is evaluated individually and assigned to one of several result paths.
- Result paths have a defined execution order and are executed sequentially. For example:
  - Decision: “True” path executes before “False” path.
  - Route: Paths execute in the order defined, with “Default” last.
  - Business Rules – Can operate the child node level of documents
  - Try/Catch – Differs in that the “Catch” path is only executed if a document failure occurs in a subsequent step.
  - Find Changes – Differs slightly in that one document enters and is internally split and assigned to CREATE/UPDATE/DELETE paths.

Decision Execution Path Example



Branch Shape

Branch shapes send a copy of the same document to every path

Behavior

- Sends a copy of the same document to every path.
- Paths are executed sequentially based on their auto-numbering. For example, “Branch 1” completes before “Branch 2” begins.
- Changes to the document (document contents, splitting/combining, document properties) made on one path are not reflected on subsequent paths. Each path gets a copy of the document in the state in which it reached the Branch shape.

Connector Shapes

Connector shapes modify document contents and/or flow by introducing documents into the flow or sending them out of the flow

Behavior

- "Get" Connector shape:
  - Introduce new documents into the process via the Start step.
  - Introduce additional documents mid-process for a given document.
  - Note: some connectors auto-split for convenience.
- "Send" Connector shapes:
  - May or may not return documents. For example:
    - Web Services SOAP Client connector returns a document with the contents of the response message.
    - Disk and FTP connectors do not return result documents, so no shapes will execute after the connector shape. They are "end of the line" shapes.
    - Note: some connectors auto-batch for convenience.

Data Modification Shapes

Data modification shapes change the data contents of documents

Shape types

- Map, Message, Load from Cache

Behavior

- Alter the data contents of a document, generally one-to-one.
- Map shapes modify data by translating from one profile to another.
  - Handle documents with multiple records by looping through records based on profile definition.
  - Can implicitly split documents, producing multiple result documents from a single input document based upon profile looping settings.
- Message shapes modify data by replacing with unstructured text (no profile).
  - Has option to combine messages resulting in a single result document.
- Load from Cache shapes actually replace the current document with a previously cached document, including data and document properties.

Data Process Shape

Data Process shapes perform actions upon a document as a whole or even across the entire group of documents

Behavior

- Perform operations across entire document data irrespective of profile (Character En/Decoding, Base64 En/Decoding, PGP En/Decrypt, Search/Replace).
- Combine multiple documents into one (Combine Documents, Zip).
- Split a single document into multiples (Split Documents, Unzip).
- Custom document data and group transformations (Custom Scripting).

Process Call Shape

Process Call shapes pass documents to another process or invoke a new process execution

Behavior

- If the sub-process's Start shape is Data Passthrough, the entire document group is passed to the sub-process for further processing.
  - Data Passthrough sub-processes execute as a continuation of the calling process's flow.
- If the sub-process's Start shape is Connector/Trading Partner/No Data, a new execution of the sub-process is spawned per document.
- Process Calls to non-Data Passthrough sub-processes can be configured not to wait for the sub-process to complete.
  - Documents will immediately continue to the next step in the calling process instead of waiting for the sub-process to complete.

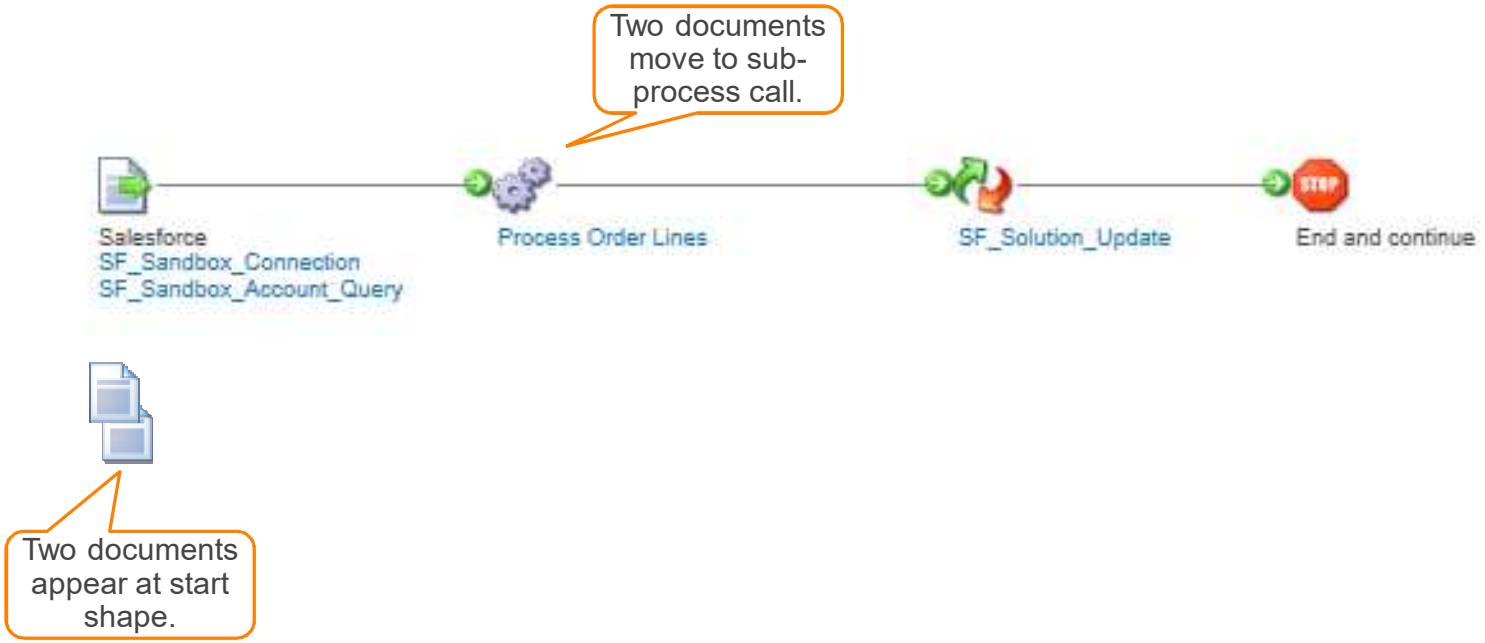
Return Documents Shape

Return Documents shapes pass documents back to a calling process

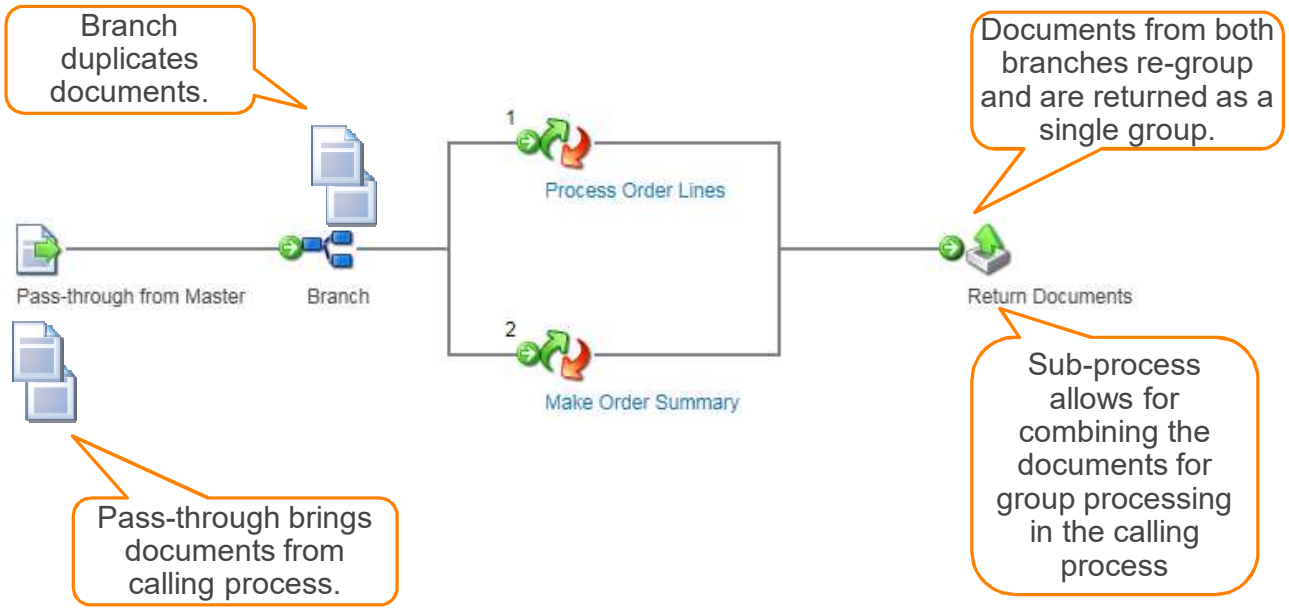
Behavior

- Pass documents from a sub-process to the calling process.
- If a sub-process is configured with multiple Return Documents shapes, multiple paths will be created from the Process Call step in the calling process (similar to a Route step).
- Waits for sub-process execution to complete before passing documents to calling process.
- “Re-group” documents from multiple paths – if multiple paths (e.g., from a Branch or Decision) connect to the same Return Documents shapes, documents from all paths are collected and returned to the calling process as single document group.
- Also used to return responses for web services listener processes.

Sub-Process Execution Path Example



Sub-Process Execution Path Example



Sub-Process Execution Path Example

Single document group is returned for further processing.

Useful when process must create additional documents for target, like batch summary records.

Path Completion Shapes

Path completion shapes are “end of the line” and do not allow for subsequent shapes to be executed on that path

Shape types

- Stop, Exception, Add to Cache, Connector

Behavior

- Stop: Terminates a given path without error and allows documents to continue on other paths. Alternatively can be configured to halt all documents immediately (rare).
- Exception: Terminates the current path with an error. Can be configured to halt the entire process (all documents) or only a single document.
- Add to Cache: Adds documents to a document cache so that they can be used in a process or subprocess.
- Connector: “Send” actions for certain connectors do not return documents (e.g., disk, FTP, SFTP, database, mail, AS2 client, JMS, Atom Queue).

Document Flow Summary

- Documents flow together (Shape execution): all documents are processed in a shape then continue on to the next shape.
- Documents flow down one path, then the next. Documents can have different paths.
- Documents can flow as a batch of logical records or as a single logical record.
- Shapes act differently on documents and batches and can modify document groupings into batches and split them out of batches.
- Branch-type shapes impact document flow. Branches process sequentially to completion and independently. Decision shape paths have precedence.
- Document failure changes execution path. Batching impacts path.


DELL
Boomi

Copyrights and Trademarks