Rules, Tasks and the API

Fundamentals of IdentityIQ Implementation

IdentityIQ

Overview

Rules, Tasks and the API

- Rules
  - What are they?
  - Where are they defined?
  - How do you interact with them?
  - Best Practices
- Tasks
  - What are tasks?
  - What are common tasks?
  - How do you interact with them?
  - Writing a custom task
- The SailPoint API
  - Overview
  - Resources
  - Common areas of usage
  - Best Practices

Rules

Rules

- Small snippets of code that can control many aspects of IdentityIQ's behavior
- Defined and stored as objects of type Rule
  - Loaded from XML
    - Exported from another IdentityIQ environment
    - Created in developers favorite IDE*
  - Created in the UI

Responsible rule development is an important skill for the IdentityIQ Implementer!

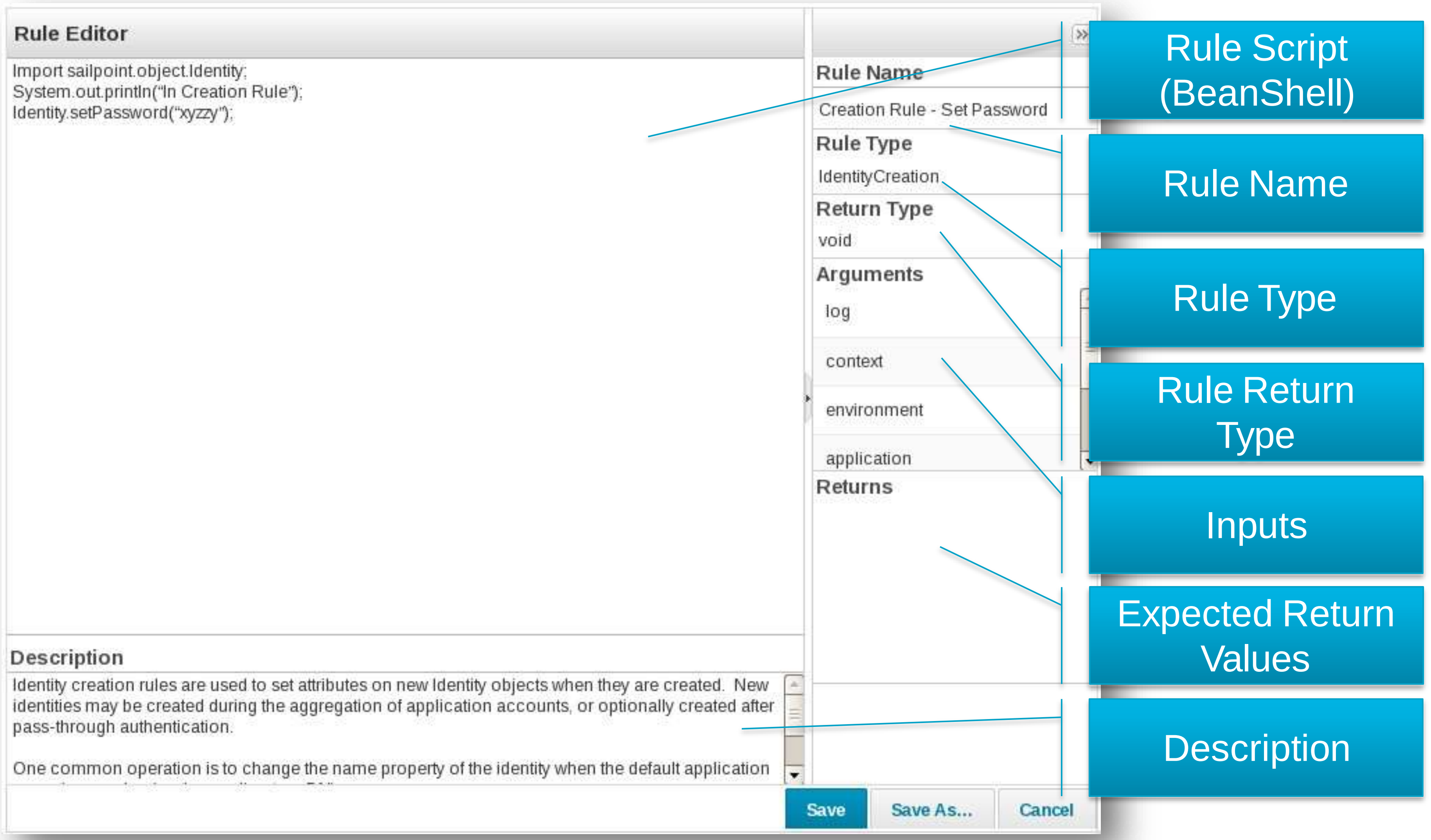
*Integrated Development Environment

Rule Usage

Why Rules?

- Control the loading of account and group data during aggregation
- Define policy violations and how to display them
- Define values, lists of allowed values and validation logic for provisioning policies and forms
- Control the behavior of certifications
- Control provisioning
- …and implement many other business goals

Example Creation Rule – Creating in UI



Rule Editor

```
Import sailpoint.object.Identity;
System.out.println("In Creation Rule");
Identity.setPassword("xyzzy");
```

Rule Name: Creation Rule - Set Password

Rule Type: IdentityCreation

Return Type: void

Arguments: log, context, environment, application

Returns

Description

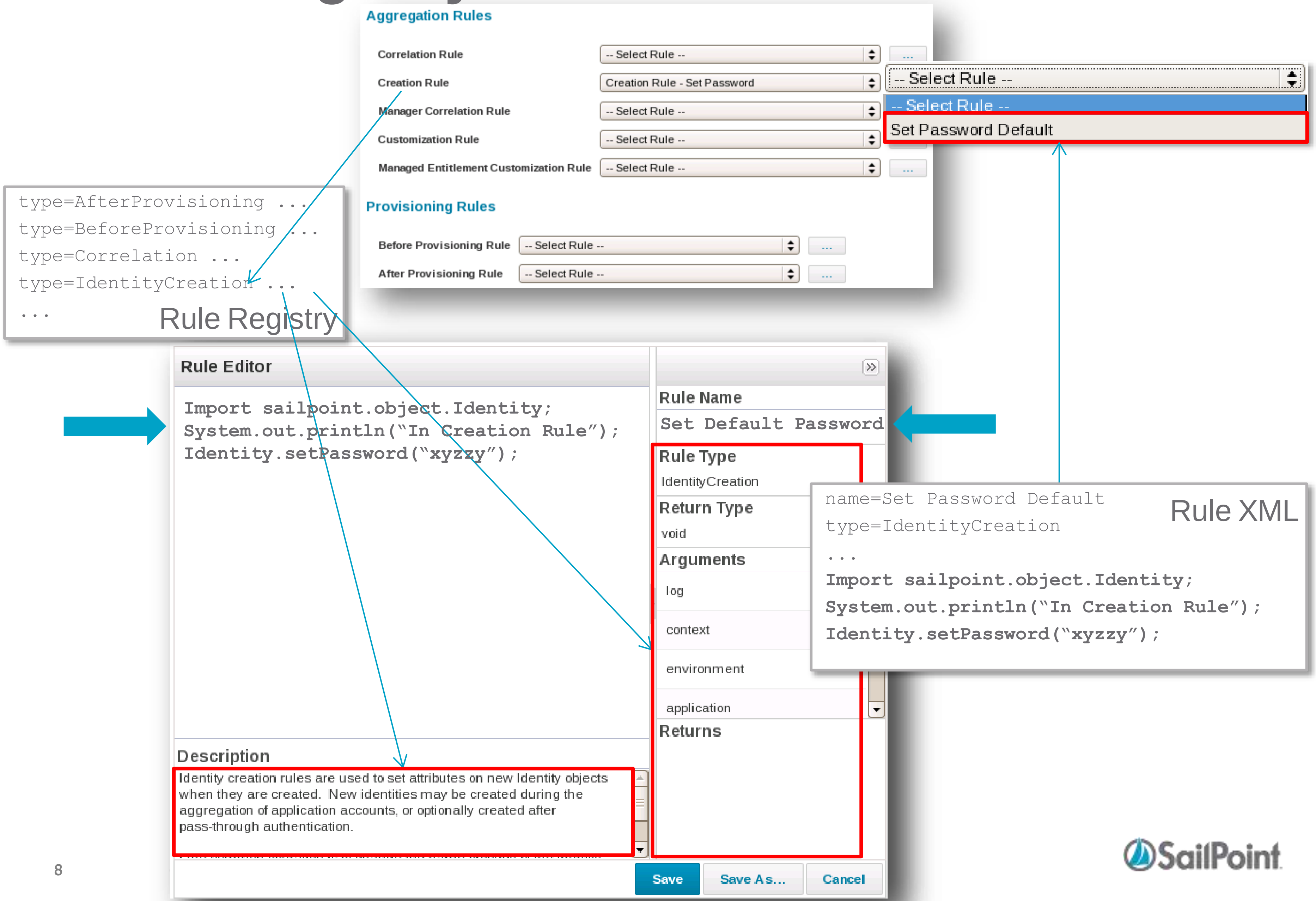
Identity creation rules are used to set attributes on new Identity objects when they are created. New identities may be created during the aggregation of application accounts, or optionally created after pass-through authentication.

One common operation is to change the name property of the identity when the default application

Anatomy of a Rule

- RuleRegistry documents each type of rule and its signature
  - Console: get RuleRegistry “Rule Registry”
  - Debug Page: select RuleRegistry object; open Rule Registry
- All rules include a type
  - Type defines where in the UI the rule can be used
- All rules are passed two objects
  - context – sailpoint.api.SailPointContext
  - log – org.apache.log4j.Logger
- All rules have inputs and most expect return values
  - Inputs/returns are defined in Rule Registry
  - Return type defines the actual Java Object being returned
    - Object, Identity, Map
  - Returns
    - List of values being returned
    - For Maps, this can be multiple entries
    - For Object, this can be one of many types of object
- Rules can set values directly or perform other actions

Rule Registry Drives Rule Creation



Aggregation Rules

Correlation Rule	-- Select Rule --
Creation Rule	Creation Rule - Set Password
Manager Correlation Rule	-- Select Rule --
Customization Rule	-- Select Rule --
Managed Entitlement Customization Rule	-- Select Rule --

Provisioning Rules

Before Provisioning Rule	-- Select Rule --
After Provisioning Rule	-- Select Rule --

```
type=AfterProvisioning ...
type=BeforeProvisioning ...
type=Correlation ...
type=IdentityCreation ...
...
```

Rule Registry

Rule Editor

```
Import sailpoint.object.Identity;
System.out.println("In Creation Rule");
Identity.setPassword("xyzzy");
```

Description

Identity creation rules are used to set attributes on new Identity objects when they are created. New identities may be created during the aggregation of application accounts, or optionally created after pass-through authentication.

Rule Name	Set Default Password
Rule Type	IdentityCreation
Return Type	void
Arguments	log, context, environment, application
Returns	

Rule XML

```
name=Set Password Default
type=IdentityCreation
...
Import sailpoint.object.Identity;
System.out.println("In Creation Rule");
Identity.setPassword("xyzzy");
```

Example Creation Rule - XML

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Rule PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<Rule created="1359045226692" id="ff8080813c6382d0013c6d6870c40226" language="beanshell" modified="1359068664579"
name="Creation Rule - Set Password Default" type="IdentityCreation">
  <Description>Identity creation rules are used to set attributes on new Identity objects when they are created …
  <Signature returnType="void">
    <Inputs>
      <Argument name="log">
        <Description>
          The log object associated with the SailPointContext.
        </Description>
      </Argument>
      <Argument name="context">
        <Description>
          A sailpoint.api.SailPointContext object that can be used to query the database if necessary.
        </Description>
      </Argument>
      <Argument name="environment" type="Map">
        <Description>
          Arguments passed to the aggregation task.
        </Description>
                 …
      </Argument>
    </Inputs>
  </Signature>
  <Source>
import sailpoint.object.Identity;
System.out.println("In Creation Rule");
identity.setPassword("xyzzy");
</Source>
</Rule>
```

Rule Name

Rule Type

Inputs standard to all rules

Inputs specific to this rule

Rule Script (BeanShell)

Rule Development

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Rule PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<Rule created="1359045226692" id="ff8080813c6382d0013c6d6870c40226" modified="1359068664579" language="beanshell"
name="Creation Rule - Set Password Default" type="IdentityCreation">
  <Description>Identity creation rules are used to set attributes on new Identity objects when they are created … </Description>
  <Signature returnType="void">
    <Inputs>
      <Argument name="log">
        <Description>
          The log object associated with the SailPointContext.
        </Description>
      </Argument>
      <Argument name="context">
        <Description>
          A sailpoint.api.SailPointContext object that can be used to query the database if necessary.
        </Description>
      </Argument>
      <Argument name="environment" type="Map">
        <Description>
          Arguments passed to the aggregation task.
        </Description>
                …
      </Argument>
    </Inputs>
  </Signature>
  <Source>
import sailpoint.object.Identity;
System.out.println("In Creation Rule");
identity.setPassword("xyzzy");
</Source>
</Rule>
```

Provided by IdentityIQ

Provided by developer

Remainder – Auto populated by Rule Editor *or* Provided by developer

Implementing Rules

- Check the signature
  - Learn the inputs
  - Learn the expected return values
  - Read the description of the rule
- Look at examples
  - Compass
  - Documentation
  - Rule Example file (/WEB-INF/config/examplerules.xml)
- General strategy
  - Figure out what you have to work with (input variables) - A
    - Can use println statements to see values being passed in
  - Figure out what you need to return (from signature) - B
  - Use API calls to get from A to B

Rules – Logging

- Logging
  - Use built in log object (log4j) for logging
    - Control logging via config file
    - No need to comment/uncomment System.out.println() messages.
  - Perform custom logging per rule

```
Logger  mylogger =
Logger.getLogger("com.xxxx.yyyy.FinanceCorrelationRule ");
mylogger.debug("This is a debug message.");
```

    - Turn it on or off

```
log4j.logger.com.xxxx.yyyy.FinanceCorrelationRule =<loglevel>
```

Rules – Performance

- Be aware of Iterative Rules
  - Rules that run many times
    - Data Loading and Correlation
      - BuildMap, MergeMaps, Transformation, ResourceObjectCustomization, Correlation
    - Certification Generation
      - Exclusion, Pre-Delegation
  - Performance of these rules can have serious impacts
    - BuildMap rule runs for every row in a 30,000 line file
    - .02 seconds * 30,000 rows = 600 seconds or 10 minutes
    - Small improvements in performance have major impact
  - Pull non-iterative functions out of iterative rules
    - Connections
    - Lookups for correlations
    - Use state or CustomGlobal to store pre-calculated information for use during iterative rules

Rules – Rule Libraries

- Create a rule containing convenience functions, etc.

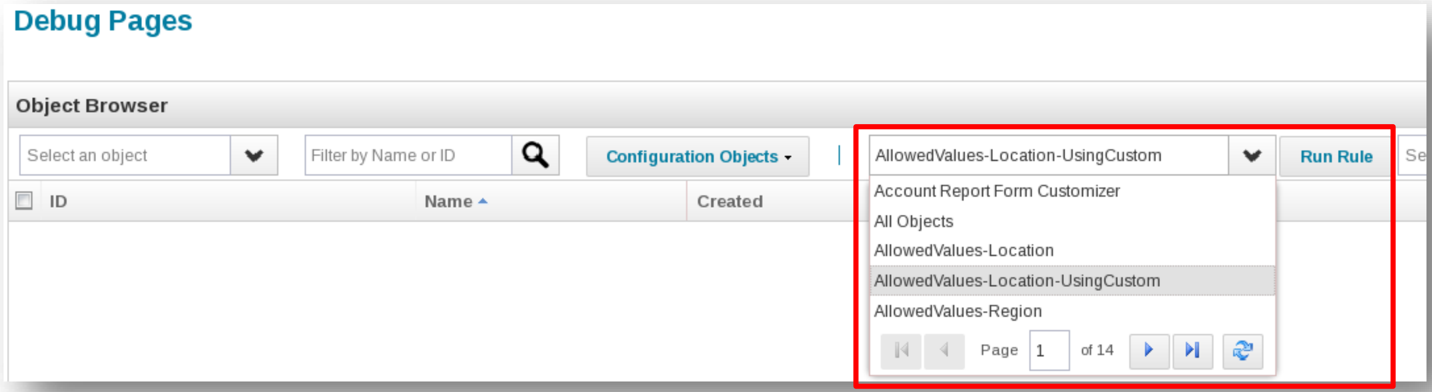
```
<Rule name='My Library'>
   <Source>
      public void doSomething() { // do stuff }
   </Source>
</Rule>
```

- Include this rule library in other rules, using:

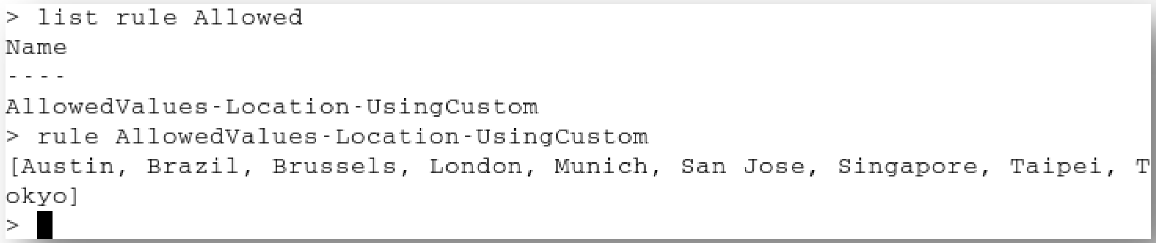
```
<Rule…>
   <ReferencedRules>
       <Reference class='Rule' name='My Library'/>
   </ReferencedRules>
   <Source>
       doSomething();
   </Source>
</Rule>
```

Rules – Executing in Debug or Console

- Debug Page
  - SailPoint context and logging objects auto-provided
  - Additional arguments must be hard coded
  - System.out sent to App Server Standard Out



- Console
  - SailPoint context and logging objects auto-provided
  - Arguments can be hard coded *or* provided in an XML file attributes map
  - System.out will go to Console



```
> list rule Allowed
Name
----
AllowedValues-Location-UsingCustom
> rule AllowedValues-Location-UsingCustom
[Austin, Brazil, Brussels, London, Munich, San Jose, Singapore, Taipei, T
okyo]
> 
```

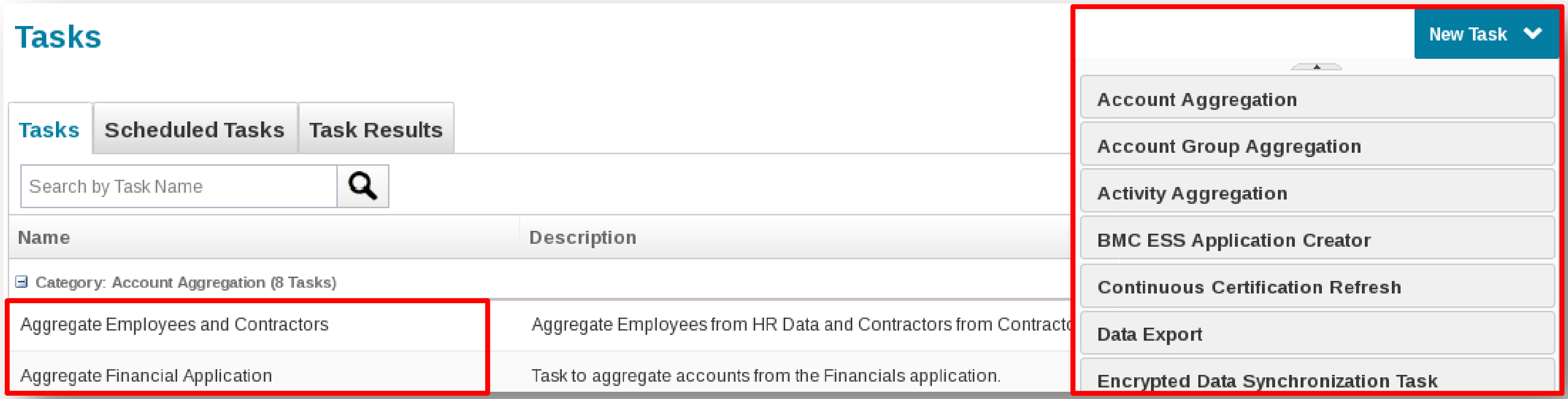
Tasks

Tasks

- Tasks perform periodic operations such as
  - Aggregation
    - Loading Accounts and Groups
  - Identity Refresh
    - Policy Violation Checking
    - Risk Scoring
    - Generating Group from Group Factories
    - Assigning and Detecting Roles
    - and More…
  - Generating LCM Text Search Index (more later)
  - System Maintenance
    - Moving Certifications along and finishing them
    - Checking for remediations
    - Pruning or archiving old objects
- Tasks are represented in the UI under Monitor → Tasks
- Tasks can be scheduled or run from the UI

Tasks – Anatomy of a Task

- Tasks can be singletons or generated using task templates
- Task templates support parameterization and creation of multiple instances of the same type of Task
  - Account Aggregation Task (Template)
  - Instances of Account Aggregation Task
    - Aggregate Employees and Contractors
    - Aggregate Financials



Tasks

Tasks | Scheduled Tasks | Task Results

Search by Task Name

Name	Description
Category: Account Aggregation (8 Tasks)	
Aggregate Employees and Contractors	Aggregate Employees from HR Data and Contractors from Contracto
Aggregate Financial Application	Task to aggregate accounts from the Financials application.

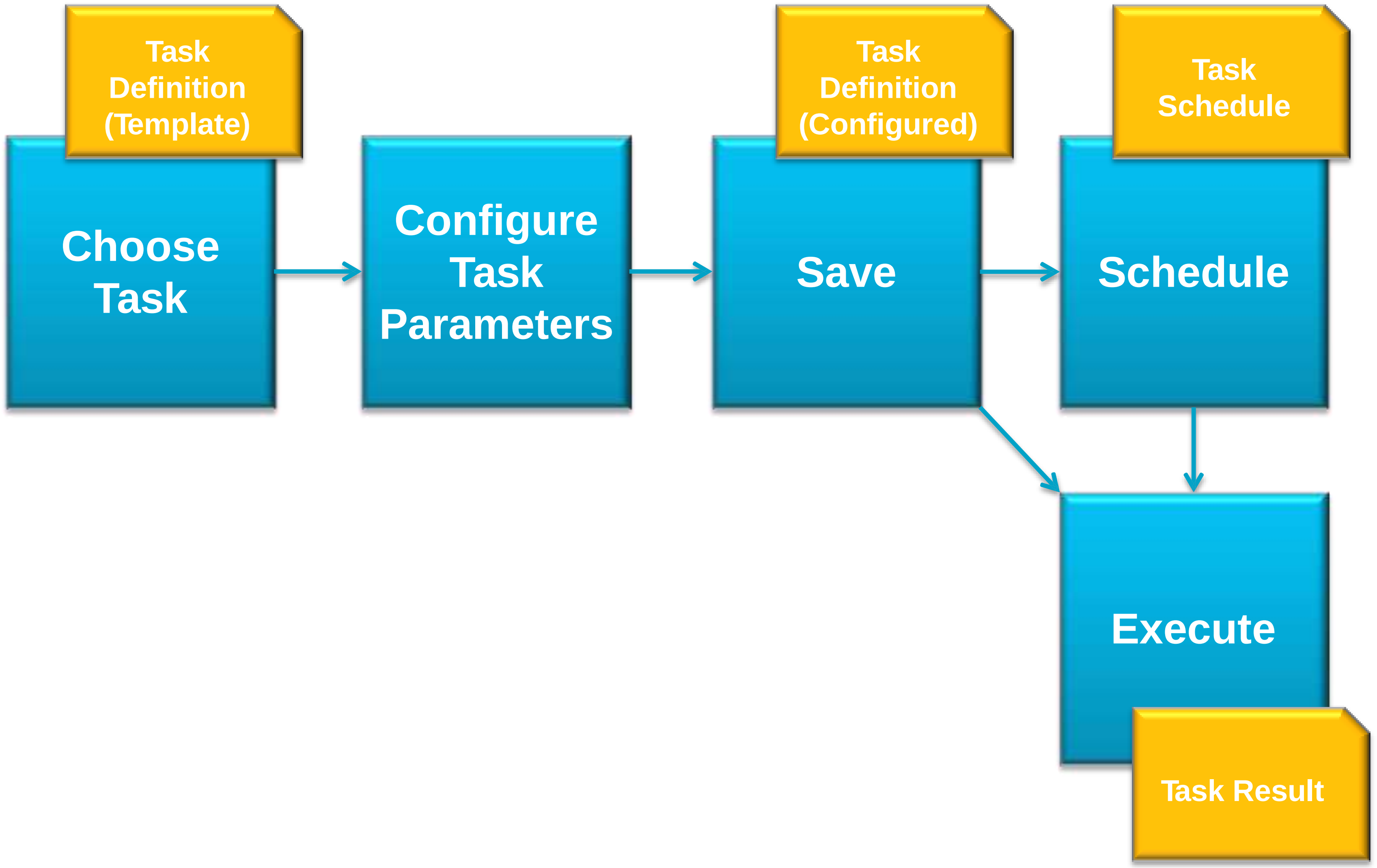
New Task

- Account Aggregation
- Account Group Aggregation
- Activity Aggregation
- BMC ESS Application Creator
- Continuous Certification Refresh
- Data Export
- Encrypted Data Synchronization Task

Tasks – Anatomy of a Task (Continued)

- Task object (XML) defines
  - Name of the task
  - Is it a template?
  - Task Signature
    - Inputs to the task
      - Applications Name
      - Checkbox reflecting task options
    - Returns from the task
      - What items will the task return once done
  - Which Java class to use to execute the task
    - All Task executors are subclassed off of Sailpoint.task.AbstractTaskExecutor

Tasks – Process and Objects


Task Definition (Template)
Choose Task
Configure Task Parameters
Task Definition (Configured)
Save
Task Schedule
Schedule
Execute
Task Result

Tasks – Creating your own

- It’s possible and very common to write your own tasks

- Method creation
  - Extend a Java class off of sailpoint.task.AbstractTaskExecutor
  - Create a TaskDefinition XML file that sets your Java class as the executor of the task.
  - Implement the following methods:

```
public void execute(SailPointContext ctx,
TaskSchedule sched, TaskResult result,
Attributes<String, Object> args) throws
GeneralException

public boolean terminate()
```

Tasks – Creating your own (continued)

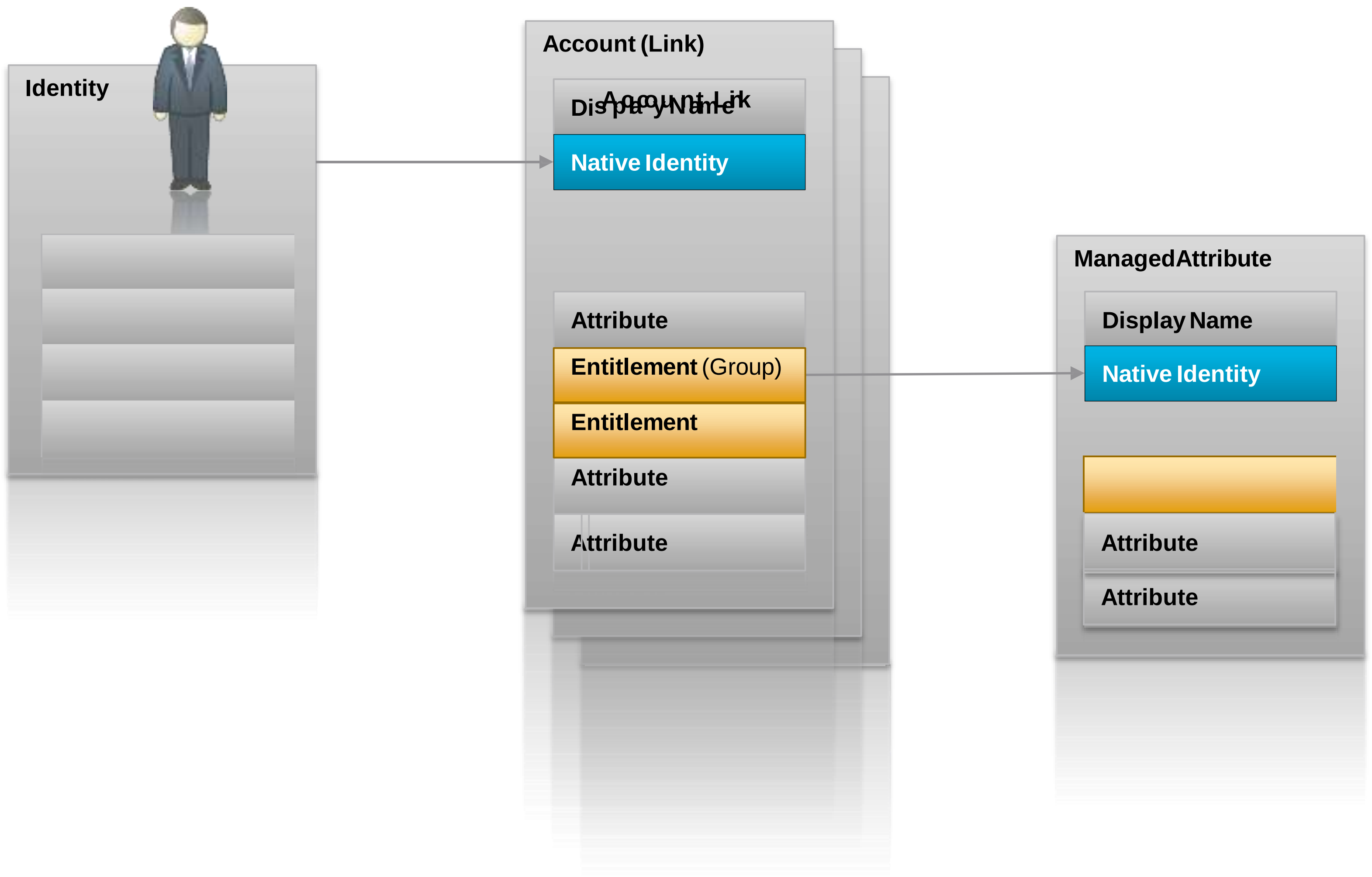
- Compile your java class and put in the classpath of your Application Server
- Load the TaskDefinition XML file
- Your task will be available to execute
  - When it runs, the execute() method of your TaskExecutor is called
  - When the task completes,
    - Results are copied into a result variable and returned
    - Results are available in the UI
- There is an example task and build/deploy environment in the training VM
  - Rule Runner Task
    - Takes a rule name as an input and allows you to run the rule at scheduled times

The SailPoint API

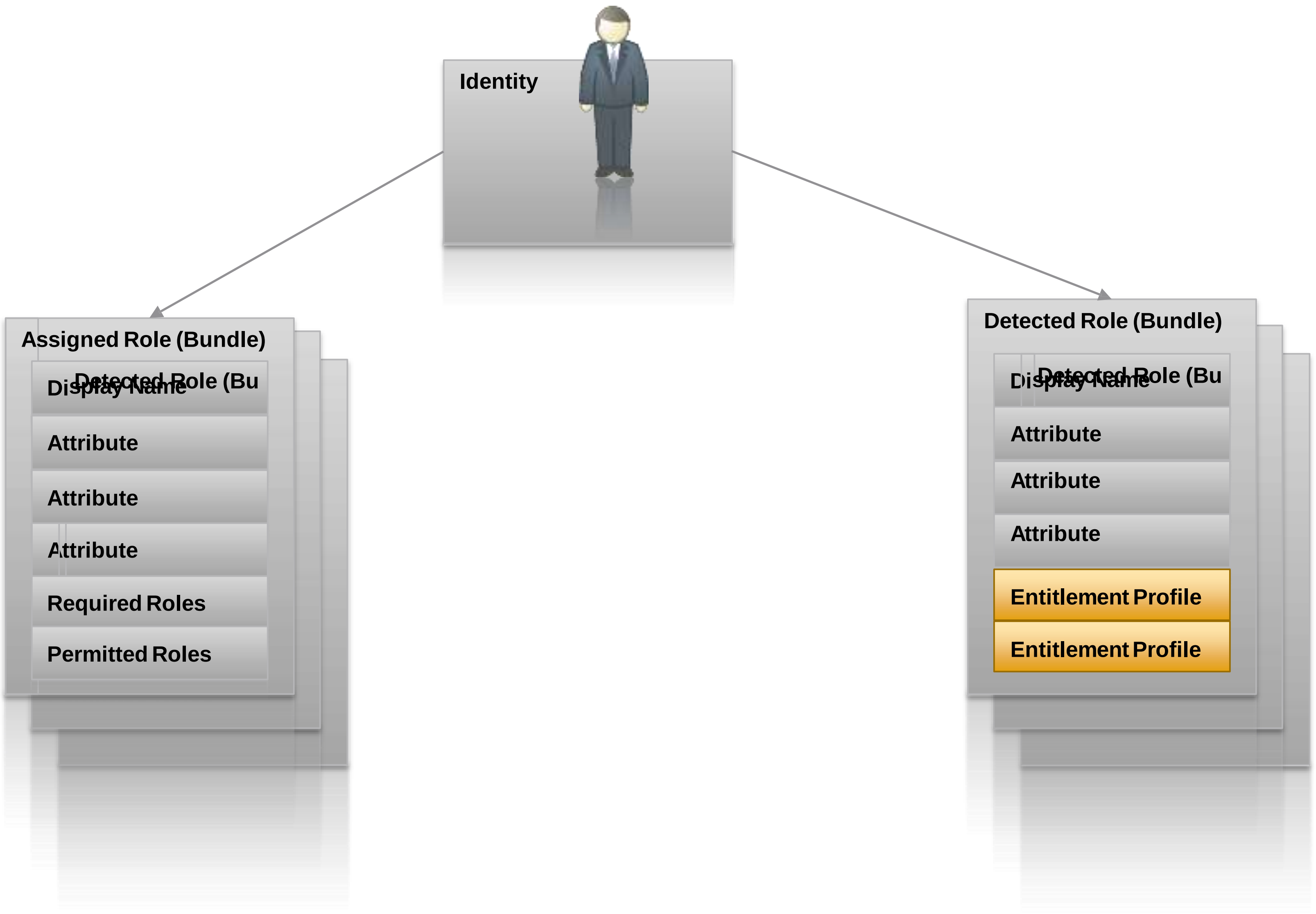
The SailPoint API

- Basic Object Model
  - Identities, Accounts and Entitlements
  - Roles
  - Certifications
  - For more, see the JavaDoc
    - in SailPoint deployment directory: /doc/javadoc/index.html
    - in training VM, click shortcut link for IdentityIQ Javadoc
- The SailPoint Context
  - Searching for objects
  - QueryOptions and Filters
  - Modifying objects
  - Saving objects

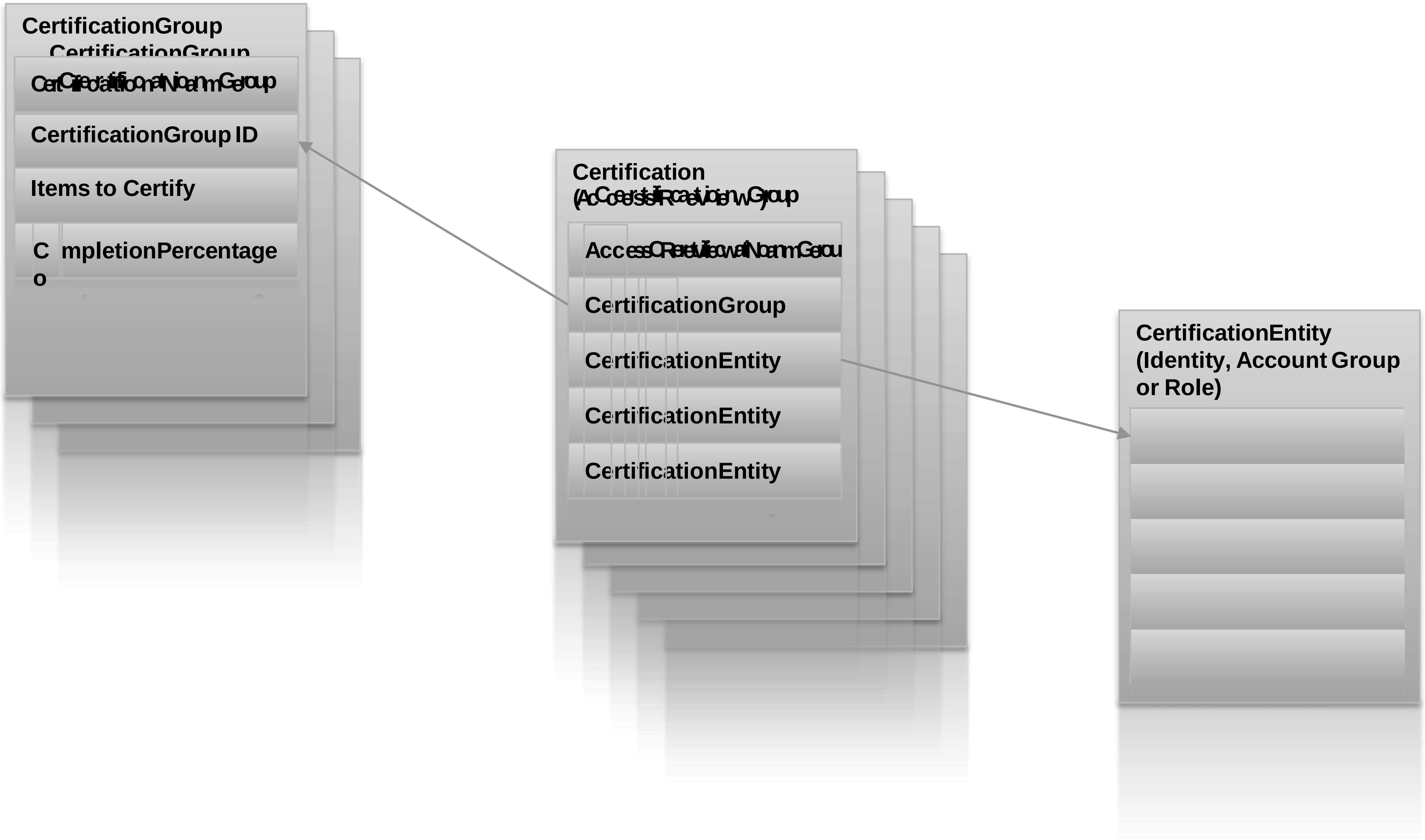
Identity/Accounts/Entitlements



Roles

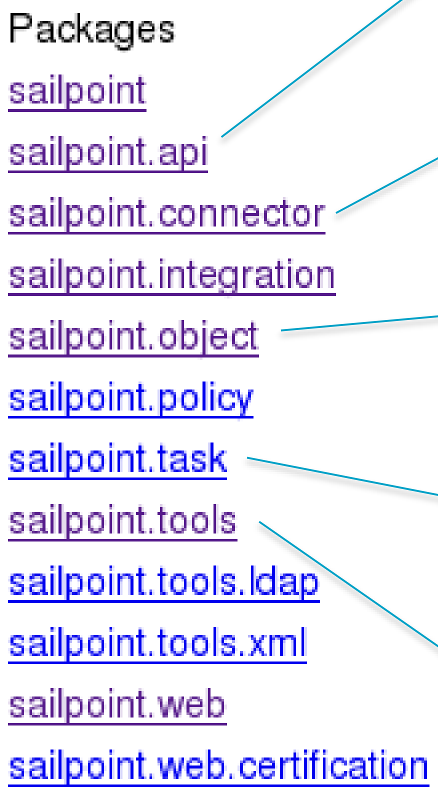

Identity
Assigned Role (Bundle)
Display Name
Detected Role (Bu
Attribute
Attribute
Attribute
Required Roles
Permitted Roles
Detected Role (Bundle)
Display Name
Detected Role (Bu
Attribute
Attribute
Attribute
Entitlement Profile
Entitlement Profile

Certifications



Object Model

- Java Doc lays out most common areas of the SailPoint object model



Packages

sailpoint

sailpoint.api

sailpoint.connector

sailpoint.integration

sailpoint.object

sailpoint.policy

sailpoint.task

sailpoint.tools

sailpoint.tools.ldap

sailpoint.tools.xml

sailpoint.web

sailpoint.web.certification

The SailPoint API Package
SailPoint Context, Emailer

Connectors
DelimitedFile, JDBC Connector, Abstract class for creating Custom Connectors

SailPoint Objects
Identity, Link, Bundle, Application, ManagedAttribute, Provisioning Plan, Provisioning Project, Filter, etc.

SailPoint Tasks
Abstract Task for creating Custom Tasks

SailPoint Tools
Util object (lots of utility methods for SailPoint development)

SailPoint *Context*

- Starting point for using the SailPoint API
- Passed into Rules, Tasks and Workflow steps
- Provides mechanisms for:
  - Determining the current logged in user
  - Getting System Configuration
  - Getting DB Connection information
  - Send an email
  - Counting items
  - Searching for items
  - Saving changes to items
  - Running rules

Finding a single Object

- To find single objects by name or Id:
  - getObjectByName(<Class>, name)
  - getObjectById(<Class>,id)

- Example (given an identity name, get their manager)

```
Identity user = (Identity)context.getObjectByName(Identity.class,”Bob.Doe”);
return user.getManager();
```

- Example (given an application id, get the name of the application)

```
Application app =
(Application)context.getObjectById(Application.class,”402881823aafe88a0
13aafe8dbfe0029”)
return app.getName();
```

Finding Objects with getObjects()

- To find multiple objects:
  - getObjects(<Class>)
  - getObjects(<Class>,queryoptions)

- Example (get all Rules)

```
List rules = context.getObjects(Rule.class);
```

getObjects() returns a java List

- Example (get all Rules of type BuildMap)

```
QueryOptions qo = new QueryOptions();
qo.addFilter(Filter.eq("type","BuildMap"));
List rules = getObjects(Rule.class,qo);
```

QueryOptions allow for the filtering of the results

Finding Objects with search()

- To find multiple objects:
  - search(<Class>,queryoptions)
  - search(<Class>,queryoptions,properties)

This type of query is a projection query

- Example (get all Identities that are uncorrelated)

```
QueryOptions qo = new QueryOptions();
qo.addFilter(Filter.eq("correlated",(Boolean)false))
Iterator identIter = context.search(Identity.class,qo);
while (identIter.hasNext()) {
    Identity identity = (Identity)identIter.next();
}
```

search() returns a Java Iterator

- Example (get the name only for each uncorrelated identity)

```
QueryOptions qo = new QueryOptions();
qo.addFilter(Filter.eq("correlated",(Boolean)false));
Iterator identIter = context.search(Identity.class,qo,"name");
while (identIter.hasNext()) {
    String identity = (String)identIter.next()[0];
System.out.println("Identity = " + identity);
```

projection search returns an array of objects (Strings)

Saving Objects

- To save objects after modifications:
  - saveObject(object)
- Example: set password on an identity

```
// assume that user is an Identity objects
user.setPassword(newPassword);
context.saveObject(user);
context.commitTransaction();
```

- Note: Many rules do not require the saving of objects that are returned from the rule.

Best Practices

- Use search() wherever possible versus getObjects()
  - search() returns a database cursor whereas getObjects returns a list of objects
- Perform filtering using QueryOptions instead of querying for all objects
- When iterating over a large volume of objects
  - use a projection query to pull in the Ids only
  - use getObjectById to get each individual object
  - occasionally (perhaps every 100 objects) call context.decache()

API by Example

- Problem: Find all uncorrelated identities in the system.
- Solution: Search for all identities, walk one by one and check the isCorrelated() method to see if they are correlated or not

```
QueryOptions qo = new QueryOptions();

Iterator result = context.search(Identity.class, qo);
while (result.hasNext()) {
    Identity user = (Identity)result.next();
    if (!user.isCorrelated()) {
        // do stuff here
    }
}
```

- Problems with this approach?

API by Example

- Problem: Find all uncorrelated identities in the system.
- Solution: Use the API to search for only the correlated Identities

```
QueryOptions qo = new QueryOptions();

// Either take a static string representation of a filter or build one
using the Filter api
//qo.addFilter(Filter.compile("correlated == false"));

qo.addFilter(Filter.eq("correlated",(Boolean)false));

Iterator result = context.search(Identity.class, qo);
while (result.hasNext()) {
    Identity user = (Identity)result.next();
    // do stuff here
}
```

- Is this better? Why?

API by Example

- Problem: Find all uncorrelated identities in the system.
- Solution: Use the API to search for only the correlated Identities' id values, then get the objects one at a time

```
QueryOptions qo = new QueryOptions();

qo.addFilter(Filter.eq("correlated",(Boolean)false));

Iterator result = context.search(Identity.class, qo,"id");

while (result.hasNext()) {
    String userId = (String)result.next()[0];
    Identity user = (Identity)context.getObjectById(Identity.class,userId);
      // Do Stuff
}
```

- Positives? Why is this better?

Questions?

Exercise Preview

Section 3, Exercises 4, 5

- Exercise 4: Using Rules to Learn the API
- Exercise 5: Compiling and Deploying a Custom Task