Rules, Tasks and the API

Fundamentals of IdentityIQ Implementation

IdentityIQ 7.0

SailPoint

Overview

Rules, Tasks and the API

- Rules
  - Defining and using
  - Best Practices
- Tasks
  - Defining and using
  - Common tasks
  - Writing a custom task
- The SailPoint API
  - Resources
  - Common areas of usage
  - Best Practices

Rules

SailPoint

Beanshell 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?

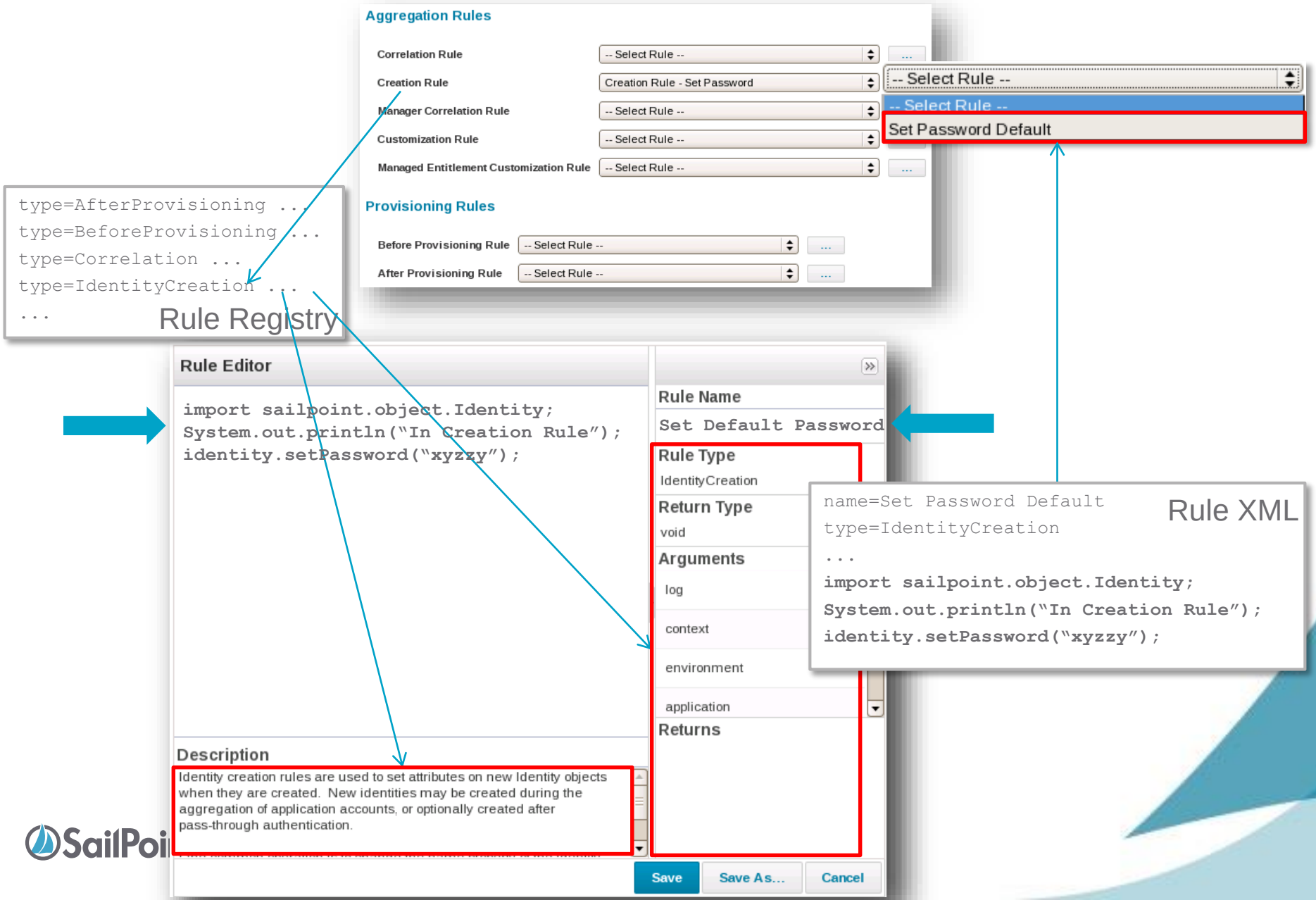
- Control the loading of account and group data during aggregation
- Define policy violations and how to display them
- Define values, lists of allowed values and validation logic for provisioning policies and forms
- Control the behavior of certifications
- Control provisioning
- …and implement many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

Anatomy of a Rule

- 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 specifies the actual Java Object being returned
    - Object, Identity, Map, etc.
  - 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

- RuleRegistry documents each type of rule and its signature
  - Console: get RuleRegistry “Rule Registry”
  - Debug Page: select RuleRegistry object; open Rule Registry

Rule Registry Drives Rule Creation



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="1359096866457
name="Creation Rule - Set Password Default" type="IdentityCreation">
  <Description>Identity creation rules are used to set attributes on new Identity objects when they are cr
  <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

Creating Rules

IdentityIQ Deployment Accelerator (optional)

- Eclipse IDE Plug-in
- Features
  - Create Rule – generates boilerplate Rule structure
  - Syntax checking for BeanShell in rules/script sections
  - Code-completion
  - Import/Export of IdentityIQ artifacts to/from multiple deployments
  - Compare local copy with deployed copy
  - Remote debug viewer
- Process
  - Download plugin from Compass (.jar file and workflow XML)
  - Place .jar file in Eclipse directory: /dropins
  - Import Workflow into each IdentityIQ system

Rules – 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
  - Code

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

  - Log4j.properties file (turn it on or off)

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

- Include log4j Trace statement every 5 lines (best practice)
  - Code

```
mylogger.trace("Entering com.xxxx.yyyy.FinanceCorrelationRule.");
```

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 object or CustomGlobal to store pre-calculated information for use during iterative rules

Rules – Rule Libraries

- Create a rule containing convenience functions, etc.

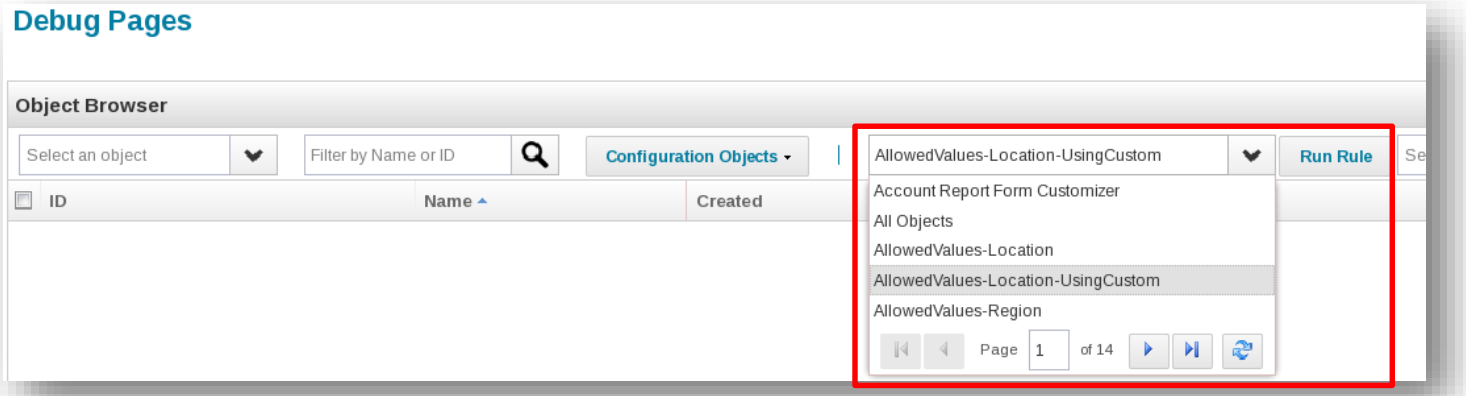
```
<Rule name='My Library'>
   <Source>
      public void doSomething() { // do stuff }
   </Source>
</Rule>
```

- 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

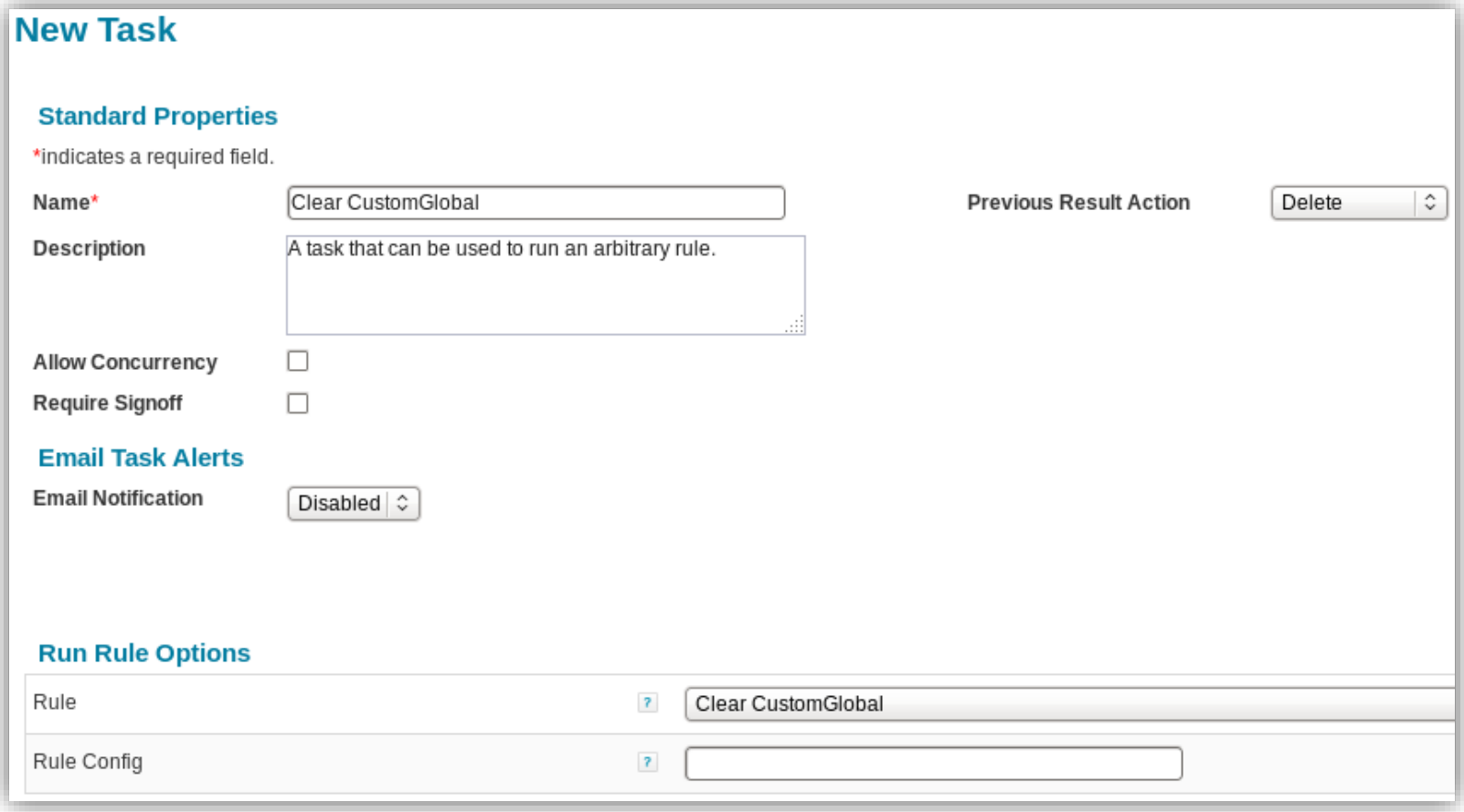


- Console
  - SailPoint context and logging objects auto-provided
  - Arguments can be hard coded *or* provided in an XML file attributes map
  - System.out will go to Console

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

Rules – Executing through Task

Run Rule Task (6.2)



Note: Rule must include code to return status

```
return "Success";
return "Failure";
```

Rules – Reference

Compass Whitepapers

- BeanShell Developers Guide for IdentityIQ (BSDG)
  - Contains examples, illustration and explanations for creating excellent code
- Rules in IdentityIQ
  - Includes descriptions, usage, and examples of all IdentityIQ rule types

Tasks

SailPoint

Tasks

- Tasks perform periodic operations such as
  - Aggregation
  - Identity Refresh
  - Running Rules
  - System Maintenance
    - Moving Certifications along and finishing them
    - Checking for remediations
    - Pruning or archiving old objects
  - Running ordered set of tasks (Sequential Task Launcher)

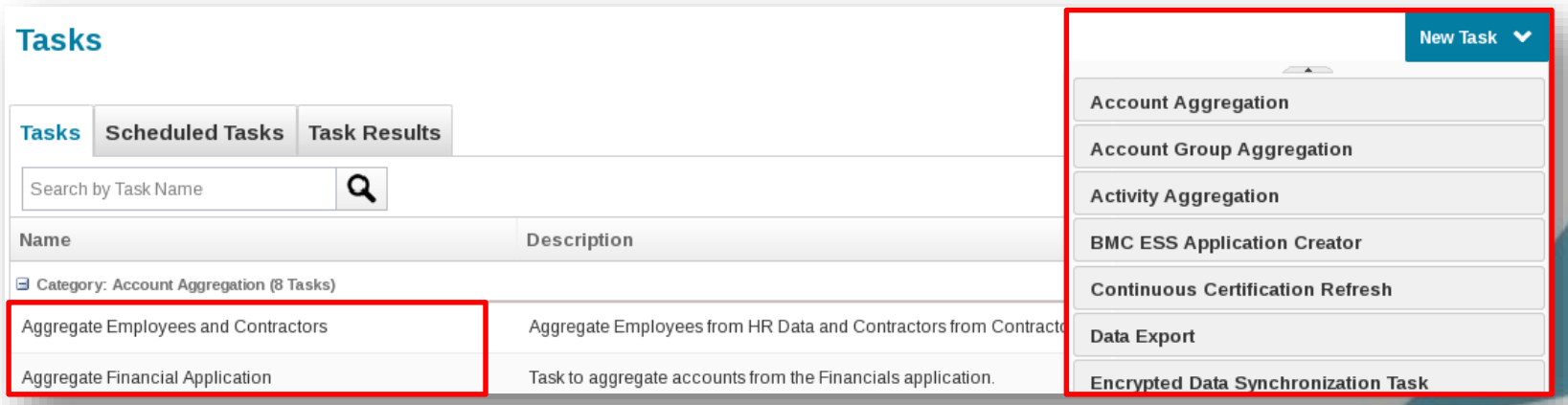
- Tasks are represented in the UI under Setup → Tasks
- Tasks can be scheduled or run from the UI
- Tasks can email upon warning/failure/always

Tasks – Anatomy of a Task

- Singleton tasks
  - Specialized purpose
  - One instance of the task can be defined per installation

  Example: Check Expired Work Items
- Template tasks
  - Support parameterization and creation of multiple instances of the same type of Task

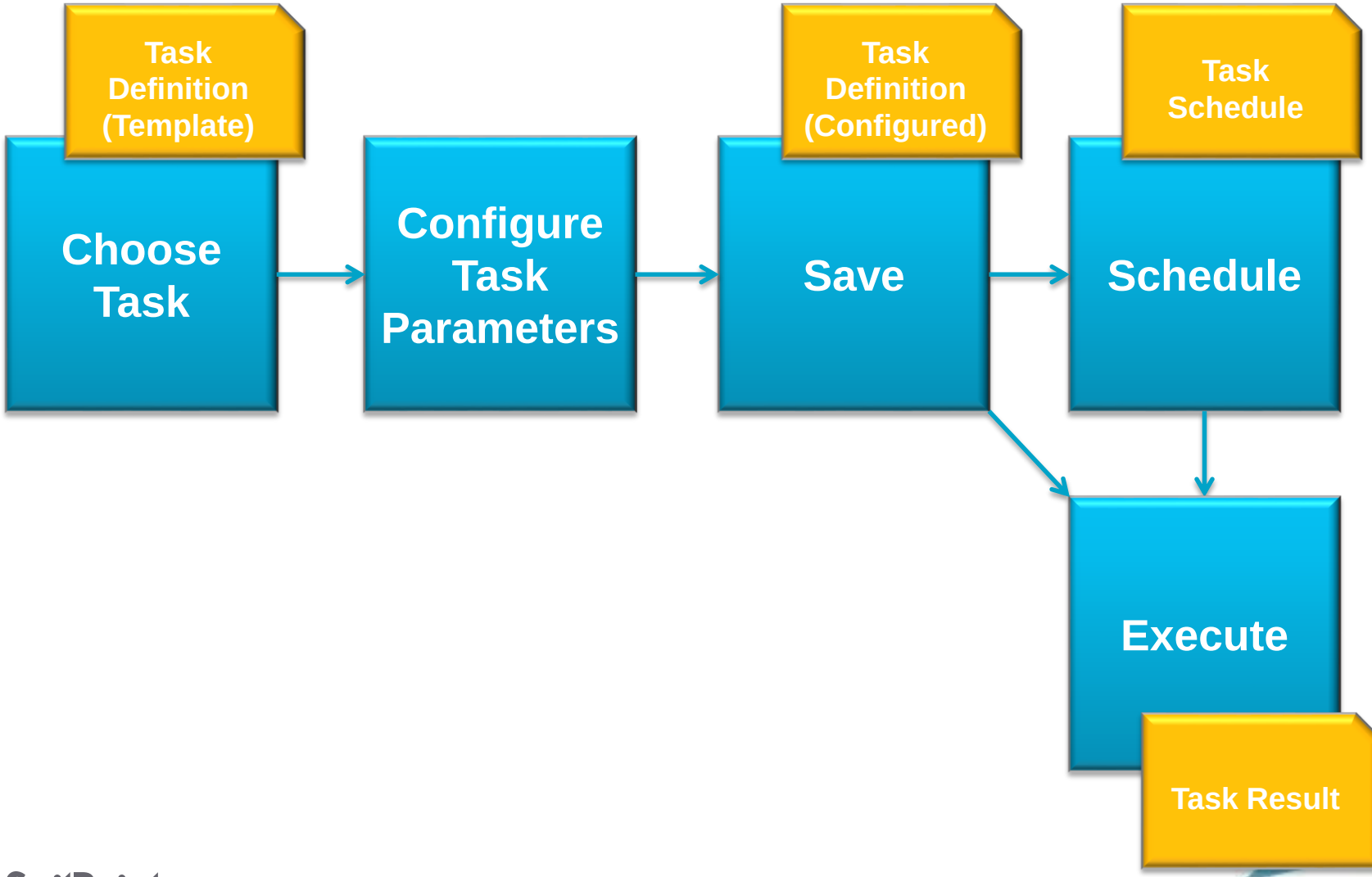
  Example: Account Aggregation



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

Tasks – Process and Objects



Tasks – Creating your own

- 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
  - Search Task
    - Searches through objects of a given class for a specified string

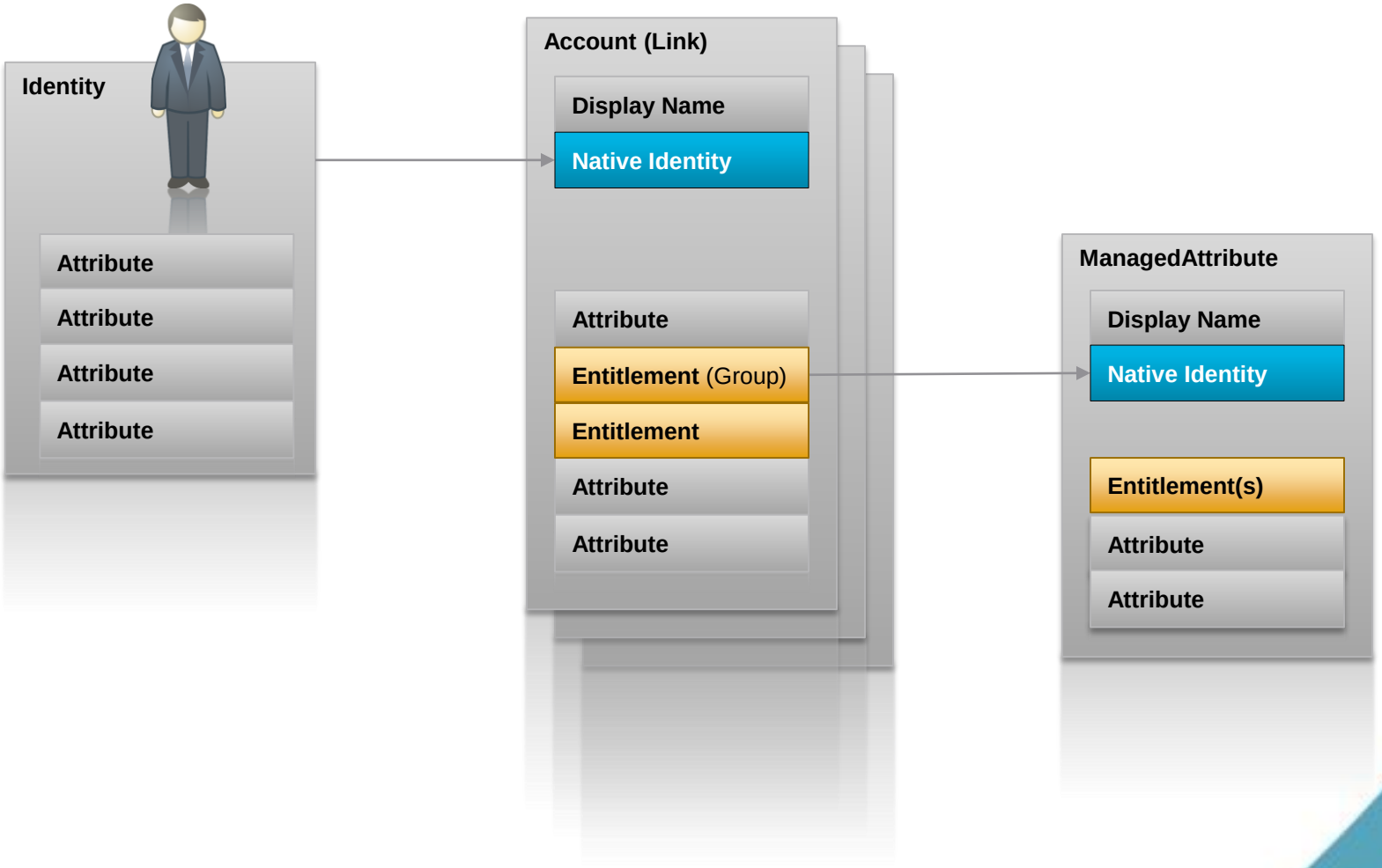
The SailPoint API

SailPoint

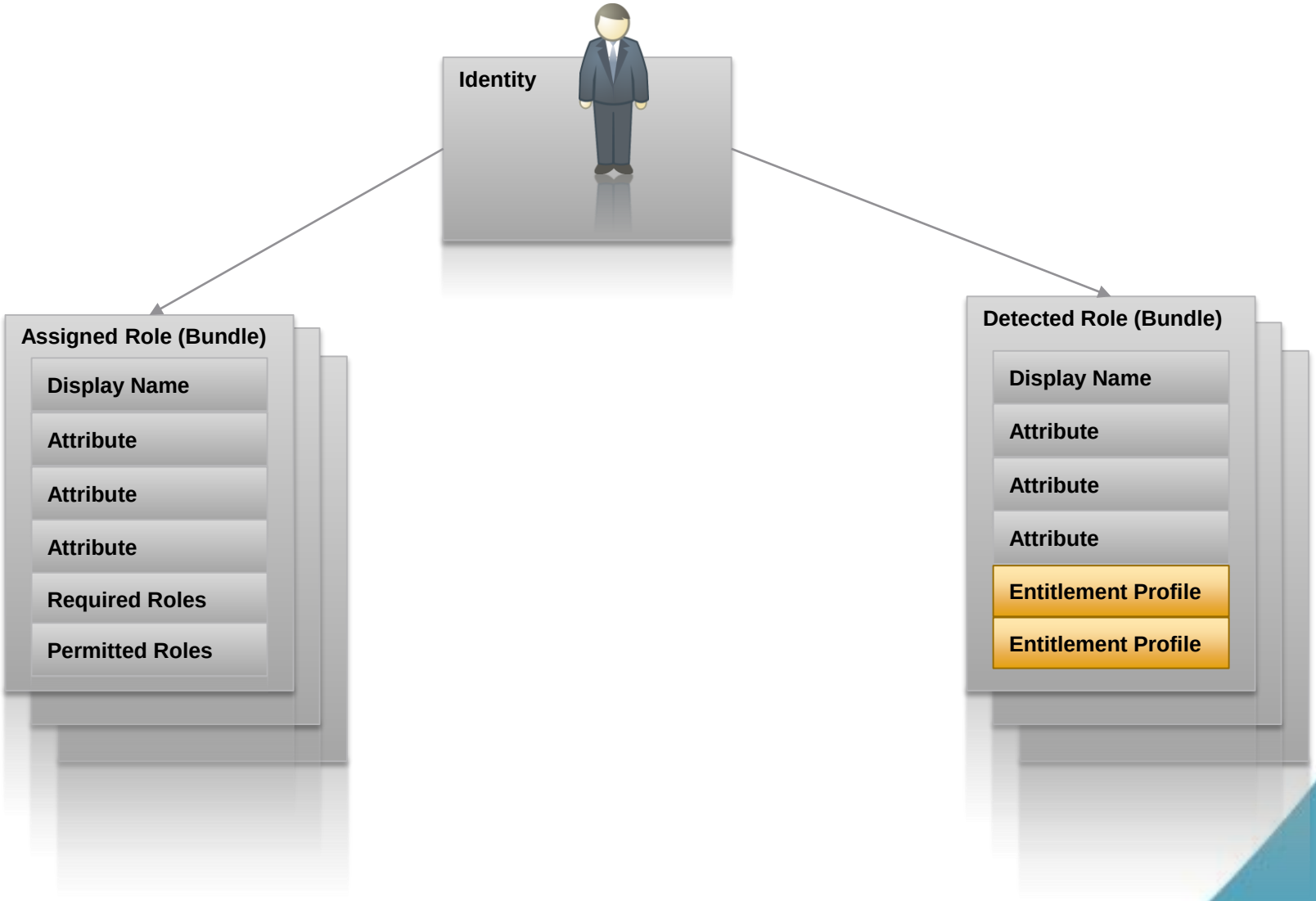
The SailPoint API

- 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

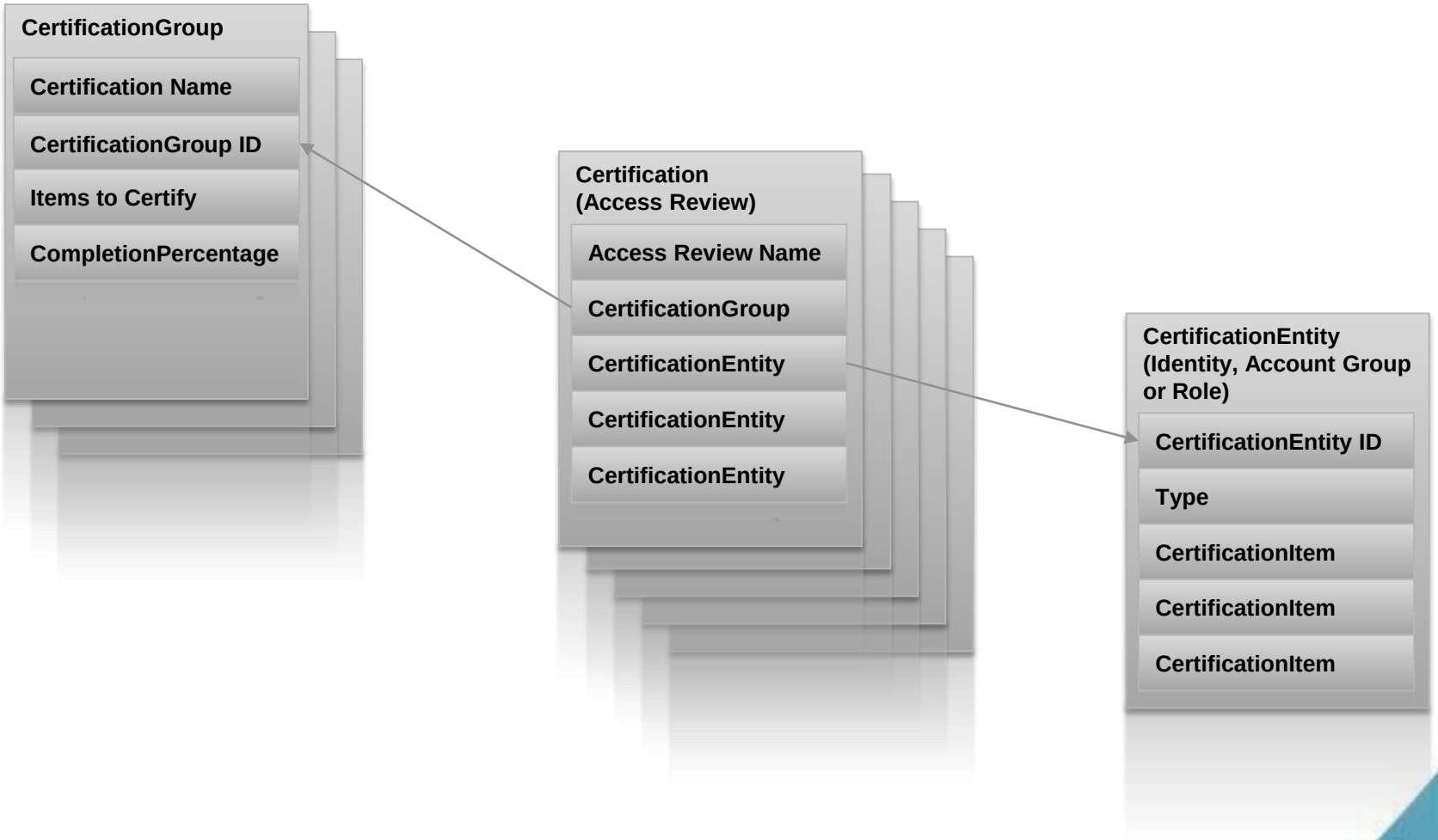
Identity/Accounts/Entitlements



Roles

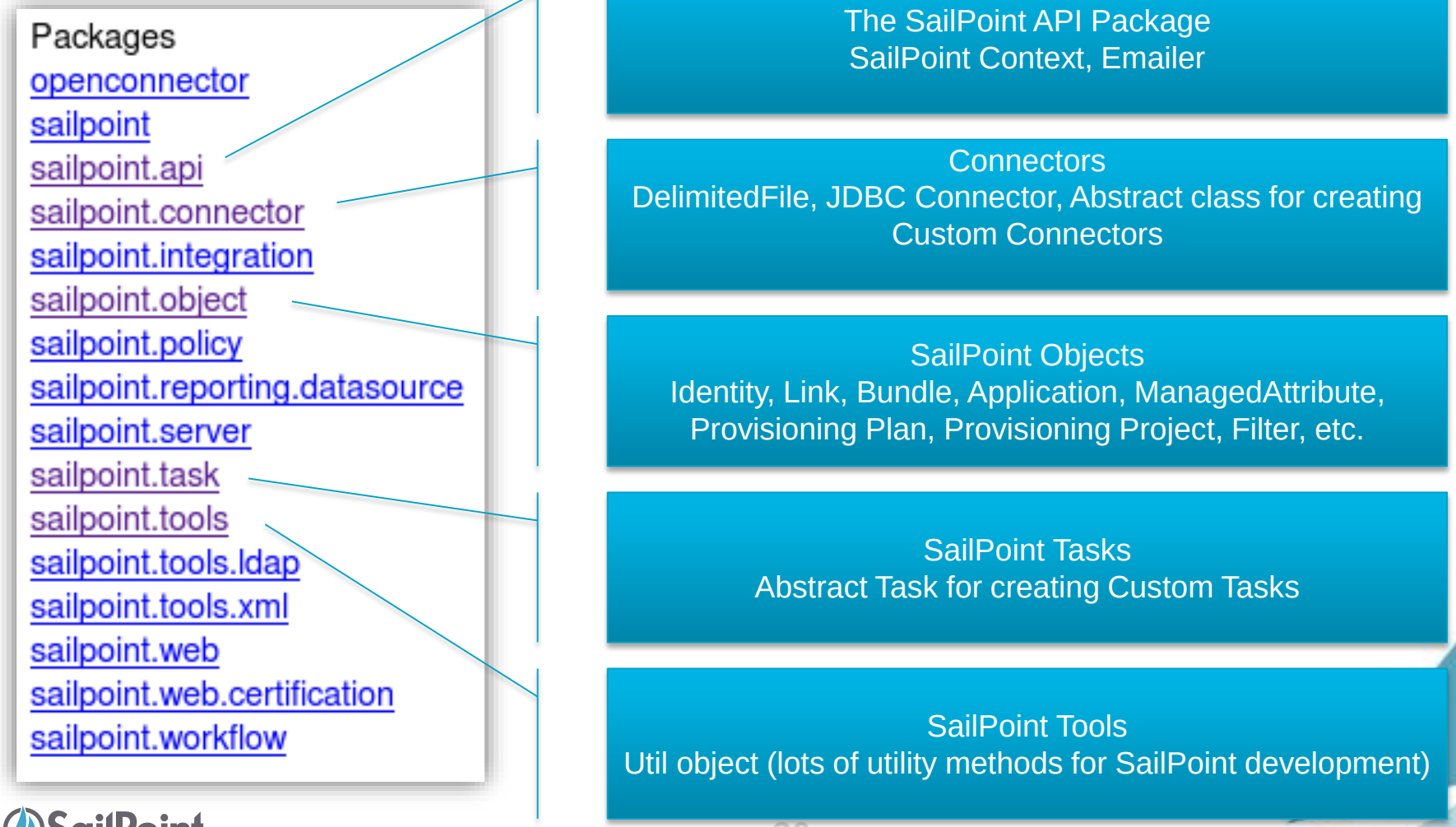


Certifications



Object Model

Java Doc lays out most common areas of SailPoint object model



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

- Find single object 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,"402881823a
afe88a013aafe8dbfe0029")
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)

Processing Cursors

- Ensure cursors are not left open
  - sailpoint.tools.Util class
  - Util.flushIterator()

- Example (look for specific account, break the loop when found)

```
QueryOptions qo = new QueryOptions();
qo.addFilter(Filter.eq("application.name","Active_Directory");
Iterator iter = context.search(Link.class, qo, "id");

while (iter.hasNext()){
    Object[] row = (Object[]) iter.next();
    String id = row[0];
    // use id of the object to fetch it and process it
   // drop out of loop
}

Util.flushIterator(iter);
```

Saving Objects

- 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() with properties 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?

SailPoint

Exercise Preview

Section 3, Exercises 4, 5

- Exercise 4: Using Rules to Learn the API
- Exercise 5: Running Tasks Sequentially and Running Rules on a Schedule
- Exercise 6: Compiling and Deploying a Custom Task