\*\*\*INSTRUCTIONAL TEXT. DELETE THIS PAGE WHEN DONE \*\*

This is a template for a functional design. A functional design is used to capture the customer’s requirements in a way that also communicates how those requirements will be met. It is laid out into functional groupings. This template attempts to cover most major use cases found in both CM and LCM projects. Validate each section carefully. Replace all yellow highlighted text with actual text and remove light blue examples.

Verstion 1.0 of Template – 10/15/2014



***<Client Name>***

*IdentityIQ*

*Functional Design*

*Phase 1*

<Author>

March 19, 2015

Version: 1.0

Table of Contents

Document Control 5

Revision History 5

Document Access Control 5

Document References 5

Document Conventions 5

Overview 6

General Configuration 7

Data Elements 7

Email Templates 7

Rights and Capabilities 7

Application Onboarding 9

PeopleSoft HR 9

Active Directory XXX 9

Application Provisioning Policies 11

Active Directory XXX 11

Entitlements and Roles 12

Entitlements 12

Organizational Roles 13

Assignable Roles 13

Birthright Roles 13

Lifecycle Events 15

Joiner Feature 16

Leaver Feature 18

Mover Feature 20

Attribute Synch Feature 22

Custom LCE Uses Provision Processor Sub 24

LCM Quick Links 26

Request Access 27

Manage Accounts 29

Secondary Account Management 30

Load/Identify Secondary Accounts 30

Secondary Account Provisioning Policies 32

Create New Secondary Account 33

Change Secondary Account Owner 35

Change Secondary Account Passwords 37

Password Management 38

Target Applications 38

Password Synch 38

Password Reset 38

Forgot Password 39

End User Change Password 39

Authentication Questions 40

Password Policy 40

Tasks 41

Custom Task 1 42

Initial Deployment Task 42

Reporting 43

Branding, Dashboard & UI 43

Branding 43

Dashboard 44

UIConfig 44

SSO and Login Configuration 45

Appendix - Services Standard Frameworks (SSF) 46

Field Value Framework 46

Dynamic Emails Framework 47

Role Assignment Framework 49

Approvals Framework 50

Provision Processor Sub Framework 51

Joiner Feature 52

Appendix - Lifecycle Events Architecture 53

# Document Control

## Revision History

| **Version** | **Author** | **Review** | **Reason For Revision** | **Date** |
| --- | --- | --- | --- | --- |
| 1.0 | <Author> | -- | Initial Draft | <Date> |

Note: This is a living document and will be subject to frequent updates as the design is directly related to other parallel phases of this project. Updates will be tracked using the Revision History table.

## Document Access Control

| **Section** | **Access Restrictions** |
| --- | --- |
| All | Restricted to project team members under NDA with <CINIT> |

## Document References

* Reference 1
* Reference 2
* SailPoint\_SSF\_UserGuide.pdf

# Document Conventions

The following conventions are being used as part of the template and/or the final document:

Explanatory text derived from the template that shouldn’t need to be changed.

Text that must be updated, completed, or removed.

Examples for text that must be updated or completed.

Text that isn’t highlighted should be carefully reviewed, but may or may not need to be changed.

Template instructions that should be removed.

**Code Example:**

ProvisioningPlan plan = new ProvisioningPlan();

logger.trace("Get the roles request");

**AccountRequest rolesReq** = **getAddOrRemoveRolesAccountRequest(context, identity);**

if (rolesReq != null){

logger.trace("Add the roles req");

plan.add(rolesReq);

}

# Overview

<Client Name> (<CINIT>) is undergoing a first phase implementation of IdentityIQ (IIQ). This phase will focus on laying a foundation and finalizing future requirements.

The full set of functionality consists of:

* SOW Item 1
* SOW Item 2

# General Configuration

There are often a number of configurations that don’t pertain to a single functional use case. Extra data tables might need to be added to store non-employee data, emails to be batched, or a record of unique identifiers. Custom mapping objects are frequently used to store non-linear data structures, email templates can be used across multiple workflows, and SP Rights and Capabilities can be added to extend the delegated administration model.

The following covers the various non-specific configurations and data elements.

## Data Elements

Various data objects and elements will be configured and/or used to solve different problems. For example, JDBC tables will be used for batch email processing. Extended identity, account, and entitlement attributes are used to drive workflow requirements. The following details the various data elements that will be used throughout the solution.

### Identity Attributes

The following identity attributes will be configured

Table - Key Cube Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Source/Details |
| joinerDate | String | Custom attribute to store the date when a joiner was completed. Used in the Identity Trigger logic for joiners. Set if successful. |

### Custom Mappings

TODO: INSERT ADDITIONAL TABLES, CUSTOM MAPPING OBJECTS, ETC.

### External Data Structures

Insert

## Email Templates

The following email templates will be developed.

Table - Email Templates

|  |  |  |  |
| --- | --- | --- | --- |
| Template | HTML Reference | Required Fields | Notes |
| <CINIT> Example Email | *<Insert any html reference for UI purposes>* | <Insert fields> | <Insert notes> |

## Rights and Capabilities

A combination of OOTB and custom rights and capabilities will be used and/or developed to facilitate delegated administration requirements, such as scoping specific quick links. Rights can be tied directly to the dynamic scope of quick links, as a means of controlling access to the link. They can then be grouped into capabilities, which are directly assigned to administrators. The following tables break down the rights and capabilities that will be in use during this phase. Any right or capability that has a name prefixed with <CINIT> is custom.

Table - SP Rights

|  |  |
| --- | --- |
| Right | Controls |
| <CINIT> Request Access | Quick Link:Request Access |
| <CINIT> Manage Accounts | Quick Link:Manage Accounts |

Table - Capabilities

|  |  |
| --- | --- |
| Capability | Rights |
| <CINIT> Helpdesk Admin | <CINIT> Edit Other Profile  <CINIT> Manage Accounts  <CINIT> Request Access  <CINIT> Password Reset  <CINIT> Create Secondary  <CINIT> Change Secondary Owner  <CINIT> Change Secondary Password |

# Application Onboarding

Applications serve to connect to a source or target system and aggregate in to IdentityIQ some or all records from the given system. One or more applications can be authoritative (a means of distinguishing orphan accounts). All applications can be configured with a correlation config or rule to link the inbound account to an existing cube; related, a creation rule can be configured to determine how to create a new cube if the inbound account was not correlated. Finally, there are a number of other rules, such as the build map or customization rules, which can be written to manipulate the inbound records.

See <CINIT>\_IIQ\_Attributes.xlsx for application schemas, identity attributes, and additional notes.

The following details the applications that are in scope and how they will be configured and used.

## PeopleSoft HR

PeopleSoft HR is the main source for all user records. It is an authoritative source and includes no entitlement information.

|  |  |  |
| --- | --- | --- |
| Question | Answer | Notes |
| Authoritative | Yes |  |
| Type | PeopleSoft |  |
| Number of Users | ~80,000 | Includes POI and inactives |
| Correlation | PERSON ID = employeeid |  |
| Entitlements | None |  |
| Aggregation Frequency | 2-3 x/day | Currently, daily. |

### Query/Filters

Insert details if there are required queries or filters

### Rules

The following rules are required for additional translation on inbound aggregations:

**Build Map**

Insert additional details.

**Identity Creation**

Insert additional details.

**<Insert additional rules>**

## Active Directory XXX

VALIDATE THE FOLLOWING: Active Directory XXX is the main account given to most users for desktop, email, and other access. Entitlements will be managed in the form of group memberships. Secondary accounts, such as shared and service accounts, will be detected and managed. Password synch will also be setup to detect native password changes to be synched to other downstream targets (see Active Directory Password Synch). Delta aggregation will be supported.

|  |  |  |
| --- | --- | --- |
| Question | Answer | Notes |
| Authoritative | No |  |
| Type | Active Directory |  |
| Number of Users | ~30,000 |  |
| Correlation | sAMAccountName = identity.name | Can also use the employeeId |
| Entitlements | memberOf |  |
| Aggregation Frequency | 1-2 hours |  |

### Query/Filters

Insert details if there are required queries or filters

### Rules

The following rules are required for additional translation on inbound aggregations:

**Build Map**

Insert additional details.

**Identity Creation**

Insert additional details.

**<Insert additional rules>**

# Application Provisioning Policies

Applications contain provisioning policies, which determine what fields get set when creating or updating an account on a given application. Each policy contains fields. Each field can be set via a default value, a script, or a rule.

Usually, fields are set via rules, specifically the rule SP Dynamic Field Value. See Field Value Framework for more information.

See <CINIT>\_IIQ\_Attributes.xlsx for the provisioning policy fields and logic and additional notes.

The following details the in scope applications, where and how each will be provisioned, and any other pertinent details.

## Active Directory XXX

Active Directory XX accounts are auto-provisioned during the joiner and rehire, and disabled during the leaver. Additional entitlements can be provisioned via Request Access. Admins can provision accounts through Manage Accounts. The security training workflows can disable and enable.

Table - AD XXX Provisioning Scenarios

|  |  |  |
| --- | --- | --- |
| Assigned | Answer | Notes |
| Joiner | Yes |  |
| Mover | No |  |
| Rehire | Yes |  |
| Leaver | Yes |  |
| Birthright Role | Yes |  |
| Requestable Entitlements | Yes |  |
| Requestable Role | No |  |
| Manage Accounts | Yes |  |
| Other Processes | Yes |  |

### Native Rules

Insert any native rule logic, such as Exchange provisioning.

# Entitlements and Roles

Entitlements are attributes that denote access on a target application. Entitlements can be loaded directly into the Entitlement Catalog, aggregated directly during a group aggregation or promoted during account aggregation.

Roles are used to bundle and assign access, usually in the form of entitlements. IT roles contain “bottom-up” entitlements that can be detected or provisioned, while business roles have “top-down” match criteria for automatic assignment and links to required IT roles. Usually, business roles can be requested manually or are assigned automatically.

The following details any entitlements that will be loaded and/or any roles that will be developed and how they will be used for birthright provisioning, request access and in future certifications.

<Insert additional details>

## Entitlements

The following details what entitlements will be configured and loaded per application.

Table - Managed Entitlements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Application | Attribute | Count | Owner | Extended Attributes | How loaded |
| App Name | Attr name | # | Attr or logic containing owner | Any additional attrs that need calculated or set | Group Agg  Promoted  Loade |
| Example:  Active Directory XXX | memberOf | ~24k | extendedAttribute1 | excludeFromEmail 🡪 (will be a new extended attribute on the group schema)  description 🡪 description  displayName 🡪 sAMAccountName by default | Group Aggregation |

### Entitlement Object Selector Scoping

By default, any manage attribute in the Entitlement Catalog can be requested. An Object Selector rule can be written to restrict certain entitlements during request access based on the requestor and requestee.

The rule will be developed with the following restrictions:

Insert requirements for entitlement object selection scoping.

## Organizational Roles

Roles can be organized hierarchically for clarity and to facilitate different operations, such as Birthright or Administrative Access.

The following organizational structure will be put into place:

Insert role hierarchy.

Example:

* **Birthright Roles** – All business roles that can be assigned during lifecycle events.
* **Assignable Roles** – All business roles that can be requested for general access. None in this phase.

## Assignable Roles

Assignable roles are requested and provisioned via the Request Access process.

The following roles will be developed for access request:

Table - Birthright IT Roles

|  |  |  |
| --- | --- | --- |
| Role Name | Inherits | Profile |
| Base IT | <CINIT> General IT Roles | Active Directory:  memberOf= “cn=DomainUsers” |

Table - Birthright Business Roles

|  |  |  |
| --- | --- | --- |
| Role Name | Match | Requires |
| Employee BR | EMP\_STATUS=”E” | Base IT |

### Role Object Selector Scoping

By default, any business role can be requested and/or assigned via the request access interface. An Object Selector rule can be written to restrict certain roles during request access based on the requestor and requestee.

The rule will be developed with the following restrictions:

Insert requirements for entitlement object selection scoping.

## Birthright Roles

Birthright roles are assigned automatically based on their match criteria. This can be done during an Identity Refresh via the options to assign/detect roles and to provision assignments, or it can be done via the Role Assignment framework in a workflow, usually an LCE workflow, such as the Joiner. See the Role Assignment framework for information on how these plans are built.

Each birthright role is a business role that has match criteria and includes one or more IT roles. The business role is assigned based on the match critieria. The IT roles include the actual access that will be provisioned.

The following roles will be built for birthright access:

Table - Birthright IT Roles

|  |  |  |
| --- | --- | --- |
| Role Name | Inherits | Profile |
| Base IT | <CINIT> General IT Roles | Active Directory:  memberOf= “cn=DomainUsers”  ou= “ou=Users,ou=POI,dc=POI,dc=<CINIT>” |

Table - Birthright Business Roles

|  |  |  |
| --- | --- | --- |
| Role Name | Match | Requires |
| Employee BR | EMP\_STATUS=”E” | Base IT |

# Lifecycle Events

Lifecycle Events are workflows that are launched, usually as a result of a refresh task. The most common Lifecycle Events are Joiner (on-boarding), Leaver (termination), and Mover (transfer). Each lifecycle event consists of a triggering mechanism and a workflow. The triggering mechanism is often a rule that evaluates the previous Identity with the new Identity and determines whether the given event should be launched. The workflow is what gets launched.

Most Lifecycle Events will use a standard architecture that relies on the SSF Provision Processor Sub Framework. Some will even use pre-built SSF Features, such as the Joiner Feature. See Appendix - Lifecycle Events Architecture for more information on the general architecture that will be used by each Lifecycle Event.

The following details the lifecycle events that will be available and/or configured.

Insert list or table to summarize events, showing quickly which ones use a feature, built on provision processor framework, or completely custom.

## Joiner Feature

Top of Form

Bottom of Form

The Joiner Feature is used to grant birthright access subsequently after the initial creation of a cube. See Joiner Feature for more details.

**The Joiner Feature will be Enabled/Disabled** (pick one, delete the other, then delete this).

If enabled, the following configurations will be made:

### Identity Trigger Logic

Identity Trigger logic determines for whom and when to launch the Joiner. The SSF Feature uses a configurable Identity Selector object to return either true (launch the workflow) or false (don’t launch the workflow).

The following logic will be configured in the Trigger Field Selector of the Custom Joiner Mapping object:

<Insert logic>

Example: If userType==”Employee” and userStatus==”Active” and assignedRoles is empty.

### Request Type/Approval Logic

A request type is a variable used to denote a use case, such as Employee Joiner or Edit Profile. It is passed into the approval framework and used to determine what approval types are required for the given use case.

The following request type logic will be configured in the getRequestType method:

<Insert logic>

Example: if userType == “Employee”, return “Employee Joiner”.

The following approval types will be required for each request type:

* Request Type 1
  + Approval Type 1
* Request Type 2
  + Approval Type 2
  + Approval Type 3

Example:

* Employee Joiner:
  + Manager

### Build Plan

The provisioning plan is the container of account requests, or what needs to be provisioned for the given user.

The provisioning plan will be built via one of the following options:

Highlight one, default is Dynamic Roles

* **Dynamic Roles** – birthright roles will be assigned based on their built-in match criteria
* **Default Assignments** – a static list of either applications or roles will be assigned
* **Custom Rule** – a custom rule will be written

If Dynamic Roles was selected above, enter the organizational role containing birthright roles:

CST Birthright Role

If Default Assignments was selected above, enter the static list of applications and/or roles.

**Applications:**

1. Active Directory
2. BR Assignment 2

**Roles:**

1. Employee Birthright Role
2. BR Assignment 2

### Custom Hook - Before Plan Logic

The following logic is required before the plan is constructed.

Insert Logic.

### Custom Hook - Before Provision Logic

The following logic is required before the project is provisioned.

Insert Logic.

### Custom Hook - After Provision Logic

The following logic is required after the project is provisioned

Insert logic.

For example:

Set the identity attribute joinerDate to the current date and time.

The following emails required. See Email Templates for details on email formatting. See Dynamic Emails for how to build the emailArgs and emailArgsList objects.

* Employee
  + To Address – new employee’s manager’s email
  + Email Template - <CINIT> New Employee Notification Email
  + Additional Attributes: First name

## Leaver Feature

Top of Form

Bottom of Form

The Leaver Feature is used to disable or remove access after the termination or leave of absence of an identity. See Leaver Feature for more details.

**The Leaver Feature will be Enabled/Disabled** (pick one, delete the other, then delete this).

If enabled, the following configurations will be made:

### Identity Trigger Logic

Identity Trigger logic determines for whom and when to launch the Leaver. The SSF Feature uses a configurable Identity Selector object to return either true (launch the workflow) or false (don’t launch the workflow).

The following logic will be configured in the Trigger Field Selector of the Custom Joiner Mapping object:

<Insert logic>

Example: If userStatus==”Terminated” and inactive == false.

### Request Type/Approval Logic

A request type is a variable used to denote a use case, such as Employee Joiner or Edit Profile. It is passed into the approval framework and used to determine what approval types are required for the given use case.

The following request type logic will be configured in the getRequestType method:

<Insert logic>

Example: if userType == “Employee”, return “Employee Leaver”.

The following approval types will be required for each request type:

* Request Type 1
  + Approval Type 1
* Request Type 2
  + Approval Type 2
  + Approval Type 3

Example:

* Employee Leaver:
  + Manager

### Build Plan

The provisioning plan is the container of account requests, or what needs to be provisioned for the given user.

The provisioning plan will be built via one of the following options:

Highlight one, default is Disable All

* **Disable All** – disables all of their current accounts
* **Delete All –** deletes all of the current accounts
* **Selective Lists** – can specify static list of accounts to be disabled and/or deleted
* **Custom Rule** – a custom rule will be written

If Selective Lists was selected above, enter the static list of applications to be disabled or deleted.

**To be disabled:**

1. Active Directory
2. BR Assignment 2

**To be deleted:**

1. Employee Birthright Role
2. BR Assignment 2

### Custom Hook - Before Plan Logic

The following logic is required before the plan is constructed.

Insert Logic.

### Custom Hook - Before Provision Logic

The following logic is required before the project is provisioned.

Insert Logic.

### Custom Hook - After Provision Logic

The following logic is required after the project is provisioned

Insert logic.

For example:

Set the identity attribute inactive to true.

The following emails required. See Email Templates for details on email formatting. See Dynamic Emails for how to build the emailArgs and emailArgsList objects.

* Employee
  + Recipient – employee’s manager’s email
  + Template - <CINIT> Terminated Employee Notification Email
  + Additional Attributes: First name

## Mover Feature

Top of Form

Bottom of Form

The Mover Feature is used to handle the transfer of an identity. This could mean re-evaluating their birthright access, updating downstream target attributes, or launching a certification. See Mover Feature for more details.

**The Mover Feature will be Enabled/Disabled** (pick one, delete the other, then delete this).

If enabled, the following configurations will be made:

### Identity Trigger Logic

Identity Trigger logic determines for whom and when to launch the Mover. The SSF Feature uses a configurable Identity Selector object to return either true (launch the workflow) or false (don’t launch the workflow).

The workflow will be triggered by one of the following options:

Highlight one, default is Compare Links

* **Compare Links** – evaluate attributes on one or more links to see if changed. Launch if one has.
* **Selector –** write role-style match criteria in Trigger Field Selector
* **Custom Rule** – a custom rule will be written

**Compare Links**

If the Trigger Type is Compare Links, the following must be configured:

Enter the mapping of application names to schema attributes that will be evaluated:

* Application 1
  + Attribute 1
  + Attribute 2

Example:

* HR
  + DEPARTMENT\_ID
  + MGR\_ID
* Contractor Table
  + COMPANY

**Selector**

If the Trigger Type is Selector, the following logic will be configured in the Trigger Field Selector of the Custom Mover Mapping object:

<Insert logic>

Example: If previousIdentity.empType != newIdentity.empType.

**Custom Rule**

If the Trigger Type is Custom Rule, enter the name of the rule that will be written and the logic that it will contain:

<insert logic>

### Request Type/Approval Logic

A request type is a variable used to denote a use case, such as Employee Joiner or Edit Profile. It is passed into the approval framework and used to determine what approval types are required for the given use case.

The following request type logic will be configured in the getRequestType method:

<Insert logic>

Example: if userType == “Employee”, return “Employee Mover”.

The following approval types will be required for each request type:

* Request Type 1
  + Approval Type 1
* Request Type 2
  + Approval Type 2
  + Approval Type 3

Example:

* Employee Mover:
  + Manager

### Build Plan

The provisioning plan is the container of account requests, or what needs to be provisioned for the given user.

The provisioning plan will be built via one or more of the following options:

Highlight all that apply, default is Review Roles):

* **Review Roles** – re-evaluate the birthright role assignments, will dynamically add and remove as necessary
* **Attribute Sych –** will re-evaluate provisioning policy logic on all target accounts and provision any updates (depends on Attribute Synch Feature)
* **Custom Rule** – a custom rule will be written

### Launch Manager Cert

Enter “true” or “false”. If “true”, a certification for the identity will automatically be generated and assigned to the identity’s manager.

### Custom Hook - Before Plan Logic

The following logic is required before the plan is constructed.

Insert Logic.

### Custom Hook - Before Provision Logic

The following logic is required before the project is provisioned.

Insert Logic.

### Custom Hook - After Provision Logic

The following logic is required after the project is provisioned

Insert logic.

For example:

Logic to build and launch a certification for the given identity.

The following emails required. See Email Templates for details on email formatting. See Dynamic Emails for how to build the emailArgs and emailArgsList objects.

* Employee
  + Recipient – employee’s manager’s email
  + Template - <CINIT> Terminated Employee Notification Email
  + Additional Attributes: First name

## Attribute Synch Feature

Top of Form

Bottom of Form

The Attribute Synch Feature is used to synchronize updates to attributes on target applications. This differs from the OOTB version, which synchs identity attributes from source to target in that it re-evaluates provisioning policies and re-synchs target attributes that might not be directly related to a cube identity attribute. See Attribute Synch Feature for more details.

**The Attribute Synch Feature will be Enabled/Disabled** (pick one, delete the other, then delete this).

If enabled, the following configurations will be made:

### Identity Trigger Logic

Identity Trigger logic determines for whom and when to launch the Attribute Synch. The SSF Feature uses a configurable Identity Selector object to return either true (launch the workflow) or false (don’t launch the workflow).

The workflow will be triggered by one of the following options:

Highlight one, default is Compare Links

* **Compare Links** – evaluate attributes on one or more links to see if changed. Launch if one has.
* **Selector –** write role-style match criteria in Trigger Field Selector
* **Custom Rule** – a custom rule will be written

**Compare Links**

If the Trigger Type is Compare Links, the following must be configured:

Enter the mapping of application names to schema attributes that will be evaluated:

* Application 1
  + Attribute 1
  + Attribute 2

Example:

* HR
  + FIRST\_NAME
  + LAST\_NAME

**Selector**

If the Trigger Type is Selector, the following logic will be configured in the Trigger Field Selector of the Custom Mover Mapping object:

<Insert logic>

Example: If previousIdentity.empType != newIdentity.empType.

**Custom Rule**

If the Trigger Type is Custom Rule, enter the name of the rule that will be written and the logic that it will contain:

<insert logic>

### No Other Triggers

An option can be set to check for other triggers and only fire this workflow if no other triggers evaluate to true. This option will be set to **true|false**. Pick one then delete the other.

### Excluded Applications

Target applications can be excluded from the process so that they’re not updated. The following applications will be excluded:

* Insert app1
* Insert app2

### Request Type/Approval Logic

A request type is a variable used to denote a use case, such as Employee Joiner or Edit Profile. It is passed into the approval framework and used to determine what approval types are required for the given use case.

The following request type logic will be configured in the getRequestType method:

<Insert logic>

Example: return “Attribute Synch”.

The following approval types will be required for each request type:

* Request Type 1
  + Approval Type 1

Example:

* Attribute Synch: (no approvals required, return an empty list)

### Build Plan

The provisioning plan is the container of account requests, or what needs to be provisioned for the given user.

The provisioning plan is built dynamically by re-evaluating the provisioning policies of each assigned application. There is only one option for how this is evaluated. However, there are options that modify its behavior

**Exclusions**

Provide the mapping of attributes per applications that should be excluded from the process:

* Example app 1
  + Attribute 1
  + Attribute 2

Example:

* Active Directory
  + password
  + userAccountControl
  + sAMAccountName

### Custom Hook - Before Plan Logic

The following logic is required before the plan is constructed.

Insert Logic.

### Custom Hook - Before Provision Logic

The following logic is required before the project is provisioned.

Insert Logic.

### Custom Hook - After Provision Logic

The following logic is required after the project is provisioned

Insert logic.

For example:

Logic to build and launch a certification for the given identity.

The following emails required. See Email Templates for details on email formatting. See Dynamic Emails for how to build the emailArgs and emailArgsList objects.

* Employee
  + Recipient – employee’s manager’s email
  + Template - <CINIT> Terminated Employee Notification Email
  + Additional Attributes: First name

## Rehire

The Rehire Feature is used to process an identity that has returned from being terminated or from a leave of absence. This could mean re-evaluating their birthright access, updating downstream target attributes, and/or enabling existing access. See Rehire Feature for more details.

**The Rehire Feature will be Enabled/Disabled** (pick one, delete the other, then delete this).

If enabled, the following configurations will be made:

### Identity Trigger Logic

Identity Trigger logic determines for whom and when to launch the Rehire.

The workflow will be triggered by one of the following options:

Highlight one, default is Selector

* **Compare Links ++** – evaluate attributes on one or more links to see if changed. Launch if one has.
* **Selector –** write role-style match criteria in Trigger Field Selector
* **Custom Rule** – a custom rule will be written

**Compare Links**

If Compare Links was selected, enter the applications and attributes that will be compared:

* Application 1
  + Attribute 1
  + Attribute 2

Example:

* HR
  + DEPARTMENT\_ID
  + MGR\_ID
* Contractor Table
  + COMPANY

**Selector**

If Trigger was selected, the following logic will be configured in the Trigger Field Selector:

<Insert logic>

Example: If previousIdentity.empType != newIdentity.empType.

**Custom Rule**

If Custom Rule was selected, the following logic will be configured in the Custom Rule:

<Insert logic>

### Request Type/Approval Logic

A request type is a variable used to denote a use case, such as Employee Joiner or Edit Profile. It is passed into the approval framework and used to determine what approval types are required for the given use case.

The following request type logic will be configured in the getRequestType method:

<Insert logic>

Example: if userType == “Employee”, return “Employee Rehire”.

The following approval types will be required for each request type:

* Request Type 1
  + Approval Type 1
* Request Type 2
  + Approval Type 2
  + Approval Type 3

Example:

* Employee Rehire: Manager

### Build Plan

The provisioning plan is the container of account requests, or what needs to be provisioned for the given user.

The provisioning plan will be built via one or more of the following options:

Highlight all that apply, default is Review Roles, Enable Accounts, Update Attributes)

* **Enable Accounts** – Will Re-enable accounts based on a pre-defined list
* **Review Roles** – re-evaluate the birthright role assignments, will dynamically add and remove as necessary
* **Update Attributes –** will re-evaluate provisioning policy logic on all target accounts and provision any updates (depends on Attribute Synch Feature)
* **Custom Rule** – a custom rule will be written

**Enable Accounts**

If Enable Accounts was selected, the following applications will be enabled if assigned:

* Application 1

**Custom Rule**

If Custom Rule was selected, the following logic will be configured in the Custom Rule:

<Insert logic>

### Custom Hook - Before Plan Logic

The following logic is required before the plan is constructed.

Insert Logic.

### Custom Hook - Before Provision Logic

The following logic is required before the project is provisioned.

Insert Logic.

### Custom Hook - After Provision Logic

The following logic is required after the project is provisioned:

Insert Logic.

## Custom LCE Uses Provision Processor Sub

Top of Form

Bottom of Form

Insert what the custom LCE Workflow does.

### Identity Trigger Logic

Identity Trigger logic determines for whom and when to launch the Joiner. The SSF Feature uses a configurable Identity Selector object to return either true (launch the workflow) or false (don’t launch the workflow).

The following logic will be configured to determine whether to trigger the workflow:

<Insert logic>

Example: If userStatus==”Terminated” and inactive == false.

### Request Type/Approval Logic

A request type is a variable used to denote a use case, such as Employee Joiner or Edit Profile. It is passed into the approval framework and used to determine what approval types are required for the given use case.

The following request type logic will be configured in the getRequestType method:

<Insert logic>

Example: if userType == “Employee”, return “Employee Leaver”.

The following approval types will be required for each request type:

* Request Type 1
  + Approval Type 1
* Request Type 2
  + Approval Type 2
  + Approval Type 3

Example:

* Employee Leaver:
  + Manager

### Build Plan

The provisioning plan is the container of account requests, or what needs to be provisioned for the given user.

The following logic will be configured to build the provisioning plan:

Insert logic

### Close Request

The following logic is required after the project is provisioned

Insert logic.

For example:

Set the identity attribute inactive to true.

The following emails required. See Email Templates for details on email formatting. See Dynamic Emails for how to build the emailArgs and emailArgsList objects.

* Employee
  + Recipient – employee’s manager’s email
  + Template - <CINIT> Terminated Employee Notification Email
  + Additional Attributes: First name

For other LCE workflows, you’re on your own. Insert them here.

# LCM Quick Links

The Lifecycle Manager dashboard ships with a number of OOTB quick links and can be extended with custom quick links. More than just a link on the dashboard, quick links are configurable objects that allow for control of scope, the action or workflow called, and other options, such as where the link is displayed on the screen and whether it can be called for others or for just the authenticated user.

The following details the LCM quick links that will be available and/or configured.

Insert list or table to summarize links, showing quickly which ones use a feature, built on provision processor framework, or completely custom.

## Request Access

Request Access is an OOTB feature that uses a shopping cart approach for requesting roles and entitlements. Entitlements are single attribute values, denoting access on target systems while roles are containers of entitlements or other roles. Generally, IT roles contain entitlements, business roles contain IT roles, and only business roles can be requested. The feature can be made available for self-service, administrative, and administrative bulk requests.

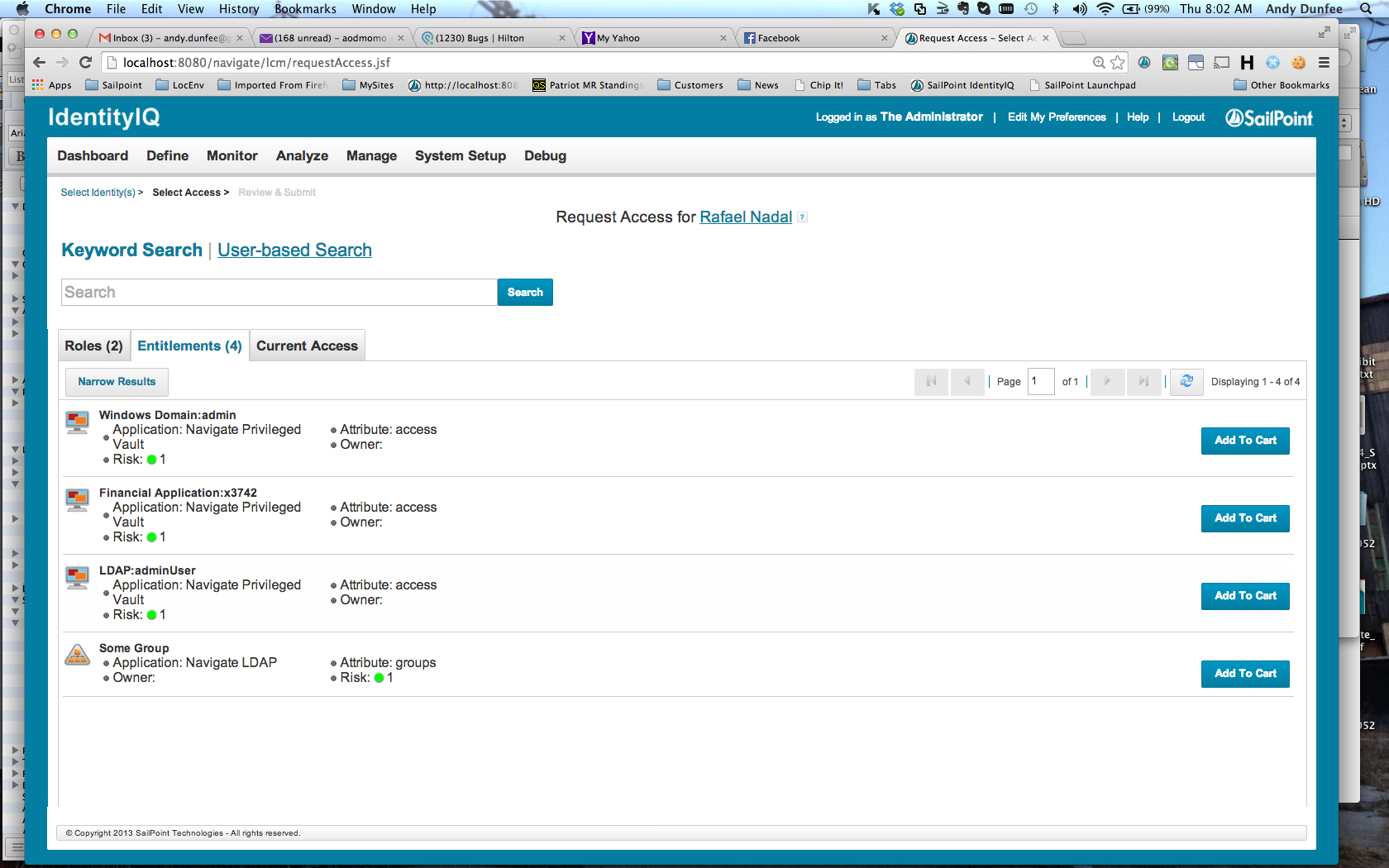


Figure - Request Access Interface

**Request Access will be Enabled/Disabled** (pick one, delete the other, then delete this).

If enabled, the following details how the feature will be used and configured:

### Link Scoping

The request access link can be restricted to certain users for both “For Me” (self-request) or “For Others” (administrative request).

The following restrictions will be put in place:

Insert Restrictions

### Entitlement Scoping

The entitlements available can be scoped per the requestor and/or requestee. Entitlement scoping details may also be described in Entitlement Object Selector Scoping.

The following restrictions will be put in place:

Insert Restrictions

### Role Scoping

The roles available can be scoped per the requestor and/or requestee. Role scoping details may also be described in Role Object Selector Scoping.

The following restrictions will be put in place:

Insert Restrictions

### Approvals

OOTB Approvals are usually done for Manager (sees all requested items) and/or Owner (approves only what they own). More complex approval scenarios should use the SSF Approvals Framework.

The following approvals are required:

Insert Approvals

### Additional Requirements

Additional requirements may be required. These may include: additional request restrictions, notifications, ticketing system integration, etc. These will need to be incorporated into the corresponding workflow.

The following are required:

Insert requirements.

## Manage Accounts

Manage Accounts is an OOTB feature that allows users and admins to create, delete, disable, enable, and unlock accounts.

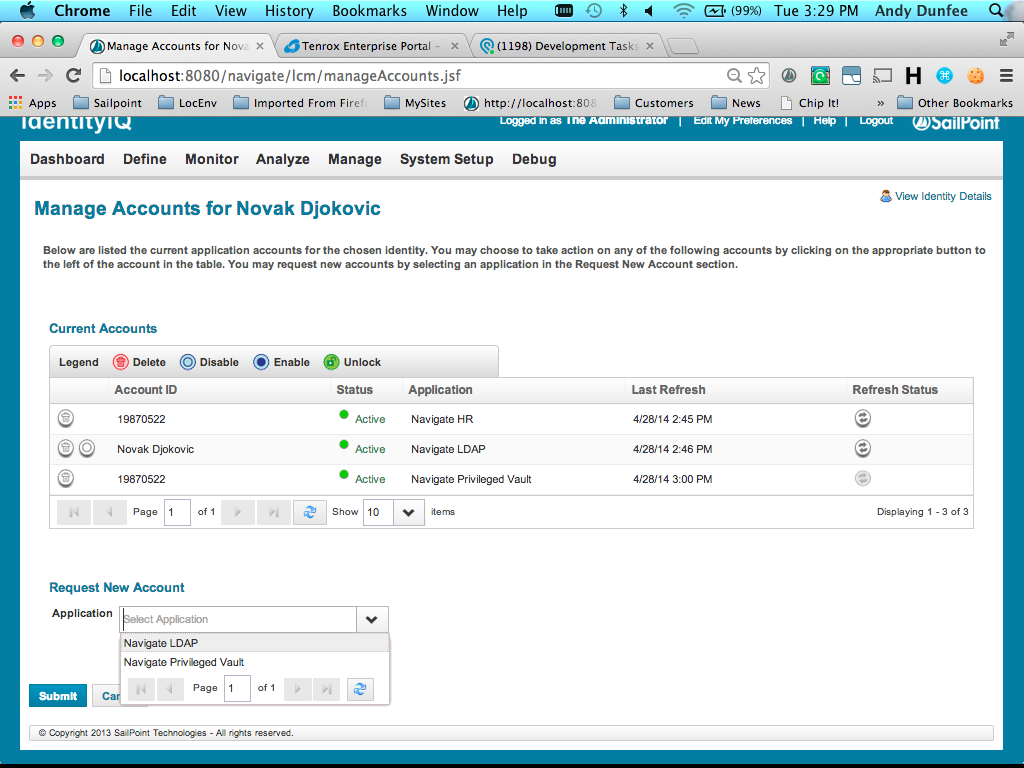


Figure - Manage Accounts Interface

**Manage Accounts will be Enabled/Disabled** (pick one, delete the other, then delete this).

If enabled, the following details how the feature will be used and configured:

### Link Scoping

The request access link can be restricted to certain users for both “For Me” (self-request) or “For Others” (administrative request).

The following restrictions will be put in place:

Insert Restrictions

### Approvals

OOTB Approvals are usually done for Manager (sees all requested items) and/or Owner (approves only what they own). More complex approval scenarios should use the SSF Approvals Framework.

The following approvals are required:

Insert Restrictions

### Additional Requirements

Additional requirements may be required. These may include: additional request restrictions, notifications, ticketing system integration, etc. These will need to be incorporated into the corresponding workflow.

The following are required:

* Insert requirements.

# Secondary Account Management

Secondary accounts are any account not directly tied to a single user—shared, service, etc. SailPoint generally recommends storing each secondary account as its own cube and having a link to the account’s owner.

Secondary Account Management includes: loading and identifying existing accounts, creation of new accounts/cubes, managing account ownership, termination of owners, and deletion of accounts.

**Secondary Account Management will be Enabled/Disabled** (pick one, delete the other, then delete this).

If enabled, the following used and configured:

## Load/Identify Secondary Accounts

Secondary Accounts include all accounts that are used by more than one user, that are generally not deleted when any one individual leaves. Each secondary account will be stored in a single, individual cube. The cube will only contain the secondary account and the name of the cube will reflect:

* It is not a user
* What application it resides on
* The account type
* The name of the account

It is recommended that a prefix be added to split out the secondary cubes on the Define Identities page. The unique name will be set in the Creation Rule for each given application. The naming scheme of the cube will be as follows:

**“ZZ:” + $appNameInitials + “:” + $accountType + “:” + $accountName**

$appNameInitials will be derived as follows:

* Active Directory – AD

$accountType and $accountName will be based on application-specific logic.

To correctly identify, name, and setup secondary accounts, the following is required for each application that supports secondary accounts:

* Correlation rule that doesn’t correlate for secondary (so as to create a new cube). There are fields on the account that will denote the type of account.
* Creation rule to change the name of the cube.
* Account attribute sources to show the type of account as well as set other pertinent flags.
* A source mapping for the identity attribute, ownerIdentity
* Source mappings for additional identity attributes.

THESE DETAILS ARE TO BE BROKEN OUT BELOW ON A PER APPLICATION BASIS.

### Active Directory Secondary Accounts

The accountType link attribute will distinguish the account from regular users. The attribute will be determine by the following logic:

Insert Logic

The following breaks down the different secondary types:

Table - Active Directory Secondary Accounts

|  |  |
| --- | --- |
| Type | Notes |
| Service Account | Enter details |

The following rules and attributes need to be configured:

#### Correlation Rule

Insert correlation logic.

#### Creation Rule

Insert the creation logic.

#### Owner Identity Attribute

Insert logic or field to calculate the owner identity.

#### Account Attributes

The following account attributes should be also be sourced:

Insert additional attributes.

## Secondary Account Provisioning Policies

Secondary Accounts often have different attribute requirements from regular user accounts. Therefore, the provisioning policy on the application should not be used.

The following details the provisioning policy requirements per application:

### AD Provisioning Policy

The following attributes will be provisioned during the creation of the secondary accounts, per account type listed below:

* Service Account
  + sAMAccountName = accountName from form (would be nice to validate uniqueness)
  + distinguishedName=”CN=sAMAccountName,owner’s ou”
  + extensionAttribute5=S

## Create New Secondary Account

A custom quick link and workflow will be developed to allow administrators to create new secondary accounts. The quick link will be scoped to administrators with the SP Right, <CINIT> Create Secondary Account. The workflow will include a dynamic form and will build a plan to create a new cube and target account.

### Quick Link

The quick link will be built with the following options:

* **Action** – workflow
* **Workflow name** – <CINIT> Create Secondary Account WF
* **forceAllowSelf** – true
* **hideAllowOthers** – true
* **Dynamic Scope** – Rights == SP Right, <CINIT> Create Secondary Account

### Workflow Steps

The workflow will have the following steps:

* **Build Form**
  + Get and set $entryForm variable to form, <CINIT> CSA Entry Form
  + Transition to Enter Form
* **Enter Form**
  + Display form, $entryForm, tied to an identityModel object that will contain the values entered on the form
  + Transition to Confirm if Approved
  + Transition to end if Rejected
* **Confirm**
  + Display a confirmation form, dynamically showing all values in the identityModel (make more static if the display doesn’t work out)
  + Provide buttons: Submit (next), Back (back), Cancel Request (cancel)
  + Transition to Build Plan on Approved (next), Enter Form on Rejected (back) and end on Cancel Request (cancel). Note: the workflow must be transient for cancel to work.
* **Build Plan**
  + Dynamically build a provisioning plan using the identity model. There should be two account requests:
    - IIQ – Set name, password, ownerIdentity and any additional identity attributes
    - Target account – Set native identity and other required attributes
* **Provision**
  + Call SP Provision Processor WF sub. This will Initialize, Approve (auto in this case), and Provision the plan
* **Close Request**
  + Update the links with any required account attributes
  + Append to the emailArgsList a map of:
    - To – Owner Identity’s email
    - Template – <CINIT> New Secondary Account Created Email
    - All other attributes in the target account request
* **Send Emails**
  + Call the SP Send Emails WF sub, passing in the emailArgsList

### Enter Form

The form, <CINIT> Create Secondary Account Form, will be developed with the following fields:

* **Application** – dynamic, refresh
* **Account Types** – dynamic, refresh, based on application, don’t display if application is null
  + AD
    - Service Account
      * Account Name: text
      * Description: text
      * Owner: identity selector
        + Filter out all inactive, all non-user accounts (excluding handful of special cubes like “Portal Users”), and where employee type is not “E” or “C”
      * Shared Account: checkbox
      * Privileged Account: checkbox
      * Exchange Enabled: checkbox

## Change Secondary Account Owner

A custom quick link and workflow will be developed to allow administrators to change the owner on secondary accounts. The quick link will be scoped to administrators with the SP Right, <CINIT> Change Secondary Account Owner, and will use the OOTB Select Identities to search for owner identities. The workflow will include a dynamic form and will update the secondary account’s owner, both on the cube and the target account.

### Quick Link

The quick link will be built with the following options:

* **Action** – workflow
* **Workflow name** – <CINIT> Change Secondary Account Owner WF
* **forceAllowOthers** – true
* **hideAllowSelf** – true
* **Dynamic Scope** – Rights == SP Right, <CINIT> Change Secondary Account Owner

### Workflow Steps

The workflow will have the following steps:

* **Build Form**
  + Get and set $entryForm variable to form, <CINIT> CSAO Entry Form
  + Transition to Enter Form
* **Enter Form**
  + Display form, $entryForm, that will return a list of the secondary account names and the new owner identity
  + Transition to Confirm if Approved
  + Transition to end if Rejected
* **Confirm**
  + Display a confirmation form, dynamically showing all values in the identityModel (make more static if the display doesn’t work out)
  + Provide buttons: Submit, Back, Cancel Request
  + Transition to Build Plan on Approved (next), Enter Form on Rejected (back) and end on Cancel Request (cancel). Note: the workflow must be transient for cancel to work.
* **Build Plans**
  + For each identity in the secondary accounts list, dynamically build a provisioning plan with two account requests:
    - IIQ – Set the new ownerIdentity
    - Target account – Set the new owner identity field
  + Add each plan to a list
  + Transition to Launch Requests
* **Launch Requests**
  + Loop the plans in the list
  + For each, launch the workflow, Process Secondary Account Owner WF, passing in the plan and identityName.
* **Close Request**
  + Update the links with any required account attributes
  + Append to the emailArgsList a map of:
    - To – New and Old Owner Identity’s email
    - Template – <CINIT> New Secondary Account Changed Email
    - List all accounts that were updated
* **Send Emails**
  + Call the SP Send Emails WF sub, passing in the emailArgsList

### Process Secondary Account Owner Update Workflow

The workflow will receive a plan and identityName. It will be a wrapper workflow to simply call the SP Provision Processor Sub.

### Enter Form

The form, <CINIT> Change Secondary Account Owner Form, will be developed with the following fields:

* **Current Accounts** – Multi-select of all owned accounts
* **Current Owner Identity** – Read Only, showing current owner
* **New Owner Identity** – Identity drop down
  + Filter out all inactive, all non-user accounts (excluding handful of special cubes like “Portal Users”), and where employee type is not “E” or “C”

## Change Secondary Account Passwords

A custom quick link and workflow will be developed to allow administrators and secondary account owners to change the password on secondary accounts. The quick link will be scoped to the secondary accounts that the given user is the Owner of or all secondary accounts if the administrator has the SP Right, <CINIT> Chance Secondary Account Password and will use the OOTB Select Identities to search for owner identities. The workflow will include a dynamic form and will update the secondary accounts’ password, both on the cube and the target account.

### Quick Link

The quick link will be built with the following options:

* **Action** – workflow
* **Workflow name** – <CINIT> Change Secondary Account Password WF
* **forceAllowOthers** – true
* **hideAllowSelf** – true
* **Dynamic Scope** – Rights == SP Right, <CINIT> Change Secondary Account Password

### Workflow Steps

The workflow will have the following steps:

* **Build Form**
  + Get and set $entryForm variable to form, <CINIT> CSAP Entry Form
  + Transition to Enter Form
* **Enter Form**
  + Display form, $entryForm, that will return a list of the secondary account names and the new password
  + Transition to Confirm if Approved
  + Transition to end if Rejected
* **Confirm**
  + Display a confirmation form, dynamically showing all values in the identityModel (make more static if the display doesn’t work out)
  + Provide buttons: Submit, Back, Cancel Request
  + Transition to Build Plan on Approved (next), Enter Form on Rejected (back) and end on Cancel Request (cancel). Note: the workflow must be transient for cancel to work.
* **Launch Requests**
  + Loop the identities in the list
  + For each, launch the workflow, <CINIT> Internal Password Intercept WF (simple, modified version of Password Intercept that doesn’t require a source), passing in the password and identityName.
* **Close Request**
  + NA
* **Send Emails**
  + Call the SP Send Emails WF sub, passing in the emailArgsList

### Enter Form

The form, <CINIT> Change Secondary Account Owner Form, will be developed with the following fields:

* **Current Accounts** – Multi-select of all owned accounts
* **New Password** – Validate conforms to the given password policy
* **Confirm Password** – Validate matches new password

# Password Management

Password Management encompasses all use cases pertaining to changing and resetting passwords, including: Active Directory password synch, admin password reset, end user change password and end user forgot password. The forgot password and password-reset features will rely on locally stored authentication questions and answers, which will be editable. Password synch will be used to pull in native password changes on Active Directory and synch to additional targets. Password Policies are used to ensure password complexity.

**Password Management will be Enabled/Disabled** (pick one, delete the other, then delete this).

If enabled, the following details how password management will be used and configured:

## Target Applications

The following applications will be targeted for password resets and changes, including passwords updated as a result of the forgot password and the password synch processes:

* Target 1
* Target 2

## Password Synch

Password interceptors will be setup on the following applications:

* Active Directory

The following customizations will be made to the Password Intercept workflow:

## Password Reset

Insert details.

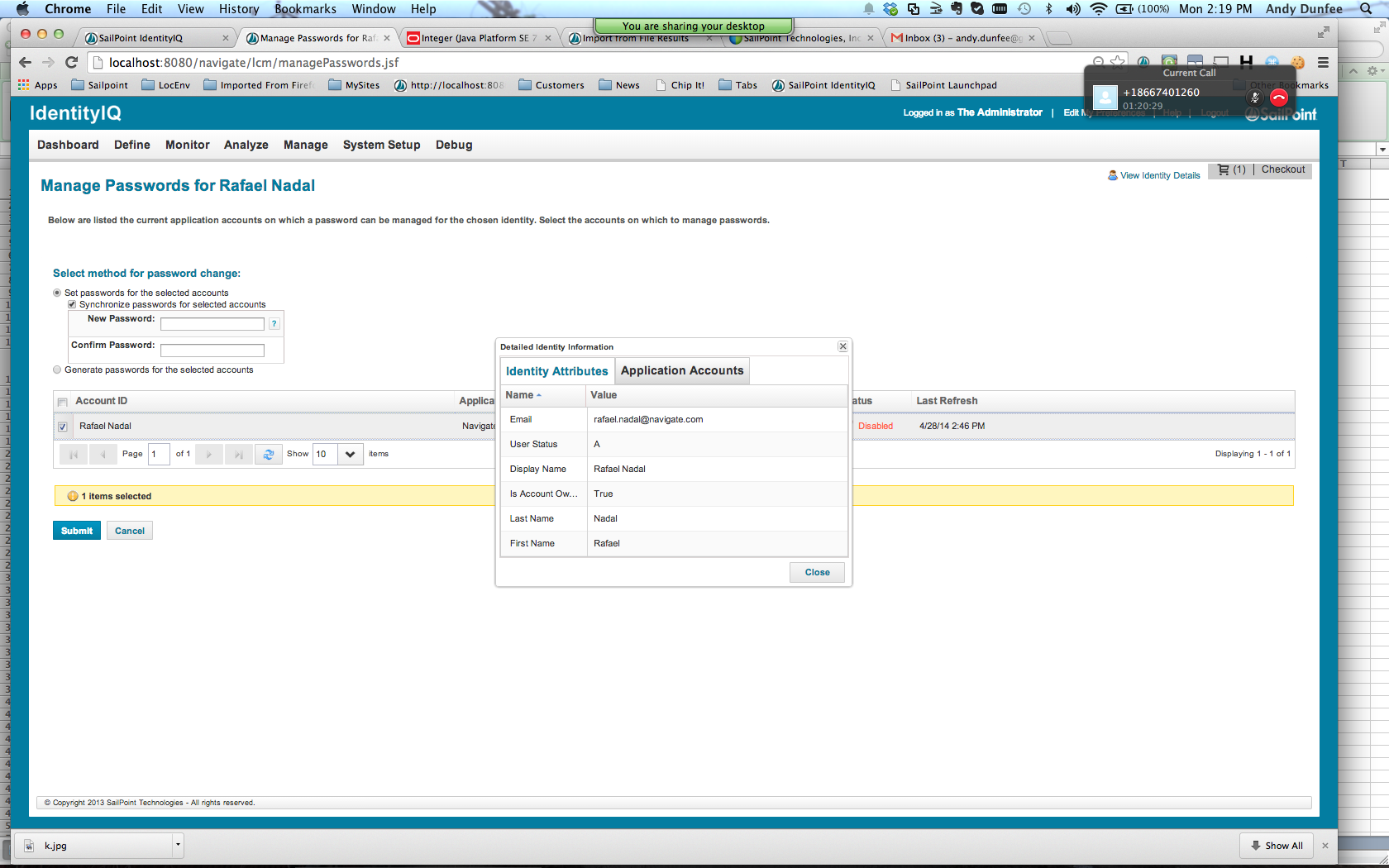


Figure - Password Reset

## Forgot Password

Insert details.

## End User Change Password

The OOTB feature will satisfy all requirements. Password generation won’t be available by default.

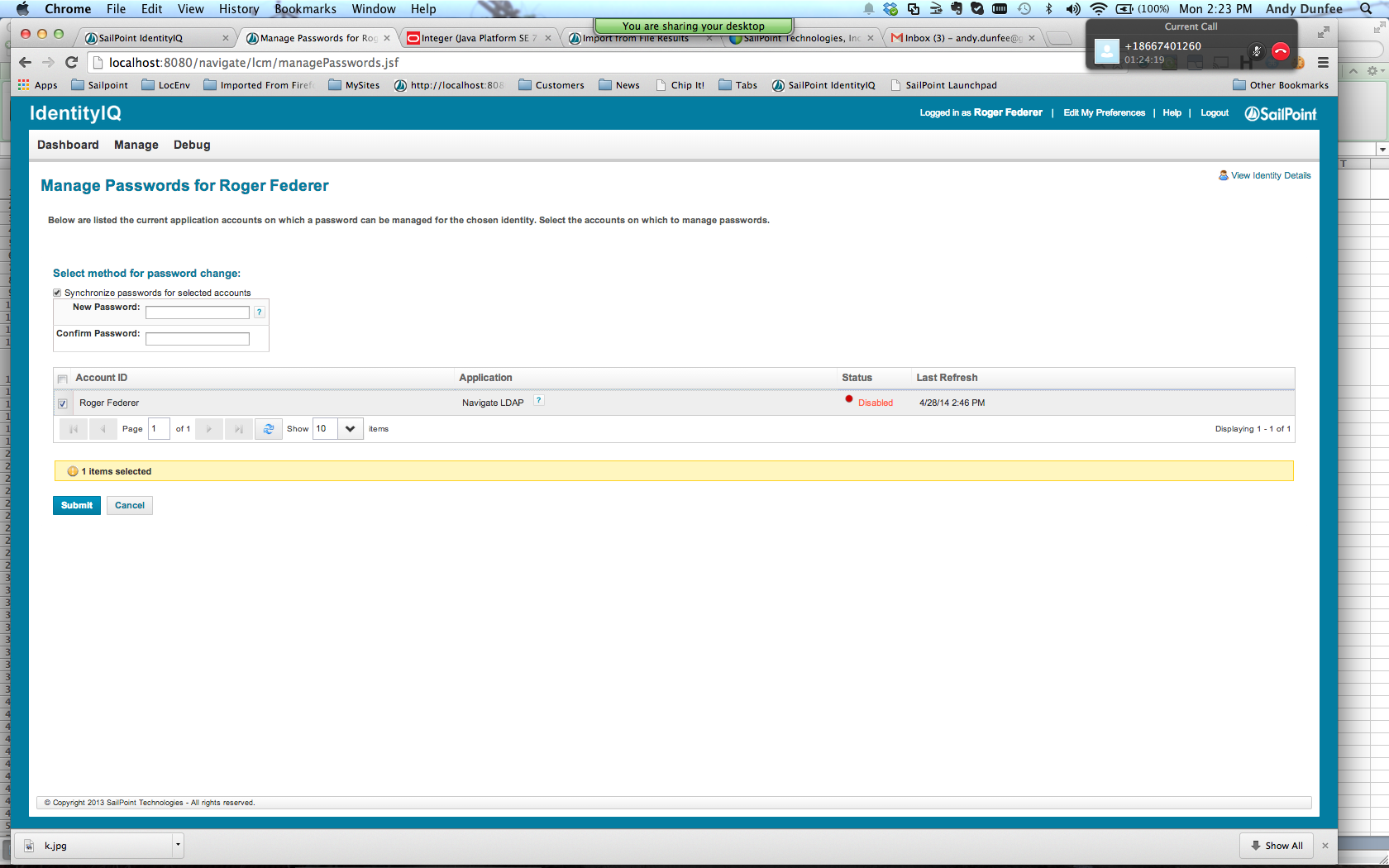


Figure - Change Password

## Authentication Questions

The following authentication questions will be used to enable the forgot password interface within IIQ. Of the 25 questions, five questions must be enrolled to use the forgot password interface. To reset a password, a user must answer at least 2 questions correctly.

Enter Questions

## Password Policy

The following details the password policy requirements:

* Use exactly eight (8) characters.
* Valid characters are letters A through Z (both lowercase and up­percase) and numbers 0 through 9.
* The first character must be a letter (either case is acceptable). \*\*
* Use at least one (1) uppercase letter.
* Use at least one (1) lowercase letter.
* Use at least one (1) number.
* Do not include your user id or a part of your name.
* Do not use common patterns of letters or numbers, such Abcd1234.
* Do not use an entire word spelled correctly.
* Check against the password dictionary, which will need to be seeded with <CINIT>-specific values.
* Cannot reuse any of the last 15 passwords

\*\* This is not currently supported with the policy and will need to be handled in the managePassword.js validate function.

# Tasks

Tasks will be scheduled at varying frequencies to aggregate accounts and groups, refresh identities, launch workflows, and do custom processing, such as nightly batch emails. The table lists the tasks that will be scheduled. Additional detail will be provided for custom tasks.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task Name | Type | Options | Frequency | Notes |
| Aggregate HR | Account Aggregation |  | 12 hours |  |
| Aggregate Active Directory Groups | Group Aggregation |  | 1 day | Aggregate both active directory instances |
| Refresh Identity Cube | Identity Refresh | Identity Attributes, Manager Status | 12 hours | Limit to just users that have anything updated, such as PeopleSoft HR |
| Refresh Detected Roles | Identity Refresh | Assign & Detect Roles | 1 week |  |
| Refresh Groups | Identity Refresh | Group Scorecards | 4 hours |  |
| Run Frequent Events Refresh | Identity Refresh | Process Events | 6 hours | Schedule to run slightly after PeopleSoft Aggregation. Limit to population of users that meet trigger criteria of events that need to run more frequently, such as Joiner and Rehire. |
| Run Nightly Events Refresh | Identity Refresh | Process Events | 1 day | Limit to population of users that meet trigger criteria of events that need to run nightly, such as Leaver, Mover, and the Training Notifications. |
| Initial Deployment Task | Custom |  | Once | See below for details. |

## Custom Task 1

A custom task will be configured to… INSERT REQUIRMENTS

## Initial Deployment Task

A custom task will be configured to do additional cube initialization prior to go live. This often includes selective assignment of existing roles; initializing identity attributes used by triggers, such as joinerDate and inactive; and other cube updates, like their existing authentication questions and answers.

Insert task requirements

# Reporting

The following reports will be configured or used.

Insert report requirements

# Branding, Dashboard & UI

The following details the various branding, dashboard and UI changes.

## Branding

Branding will be limited to images and CSS changes. For the most part, this will be a short process of trial and error, but some colors and icons can be noted.

### Screens & Examples

Insert screens

### Colors & Styles

Insert colors and other style updates

## Dashboard

The following is the OOTB dashboard with noted changes:

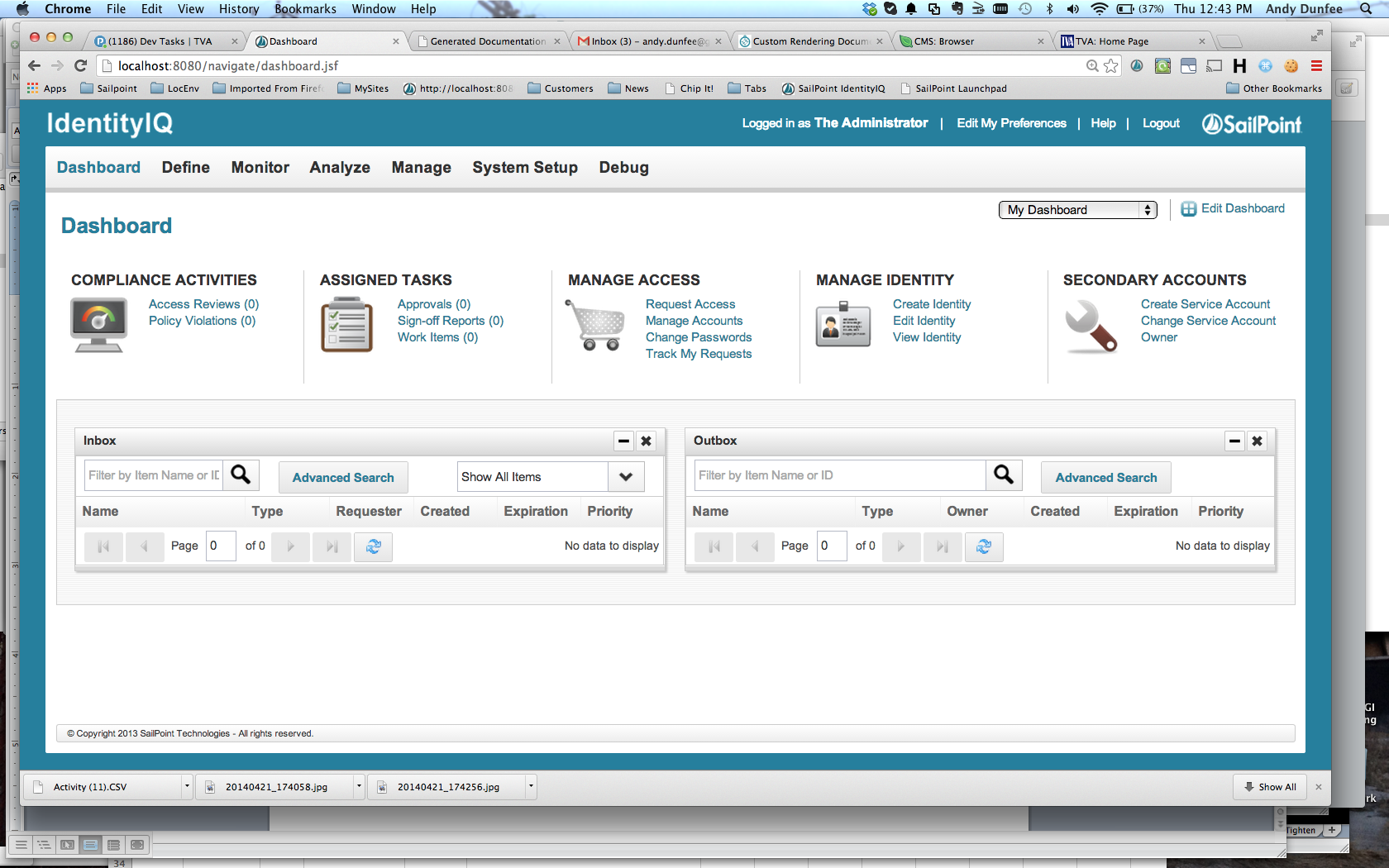


Figure - Dashboard Changes

The following changes will be made:

* Add dynamic scopes for links to appropriately show/hide
* Remove the Edit My Preferences link and add a Change Authentication Quick Link

## UIConfig

The following UIConfig changes will be made:

### identityTableColumns

Configure the columns on the Define>>Identities page. The following changes will be made:

* Add employeeid, inactive, cubeowner
* Remove Risk Score, Last Refresh Date, Assigned & Detected Role Summary

### Sailpoint.web.lcm.RequestPopulationBean

Configure the columns on the Select Identities page for any LCM request. The following changes will be made:

* Add employeeid, cubeowner

### identityViewAttributes

Configure the attributes displayed on the Attributes tab of the page displayed after clicking an identity on the Define>>Identities page.

Show every identity attribute, except for the last four of SSN.

### identitySearchAttributes

Configure the attributes that will be searched on the Define>>Identities page.

Add employeeid. Replicate search currently done on manager to ownerIdentity.

# SSO and Login Configuration

Detail the SSO and/or Login requirements.

# Appendix - Services Standard Frameworks (SSF)

The Services Standard Frameworks (SSF) will be used throughout this implementation to promote standardization, increase reusability, and decrease the overall codebase. See SailPoint\_SSF\_UserGuide.pdf for a complete breakdown of their structure, contents, and usage.

The following is a quick breakdown of how each can be used and benefit this phase of the implementation.

## Field Value Framework

The Field Value framework will be used to standardize the provisioning policies on target applications. Each field in each provisioning policy will call the single field value rule that will then direct to a common field value rule library. The naming conventions will allow for reusing field value rules for fields that are common across applications. This also allows for better use of log4j, simplification of the application definitions, and the reduction in the total number of field value rules.

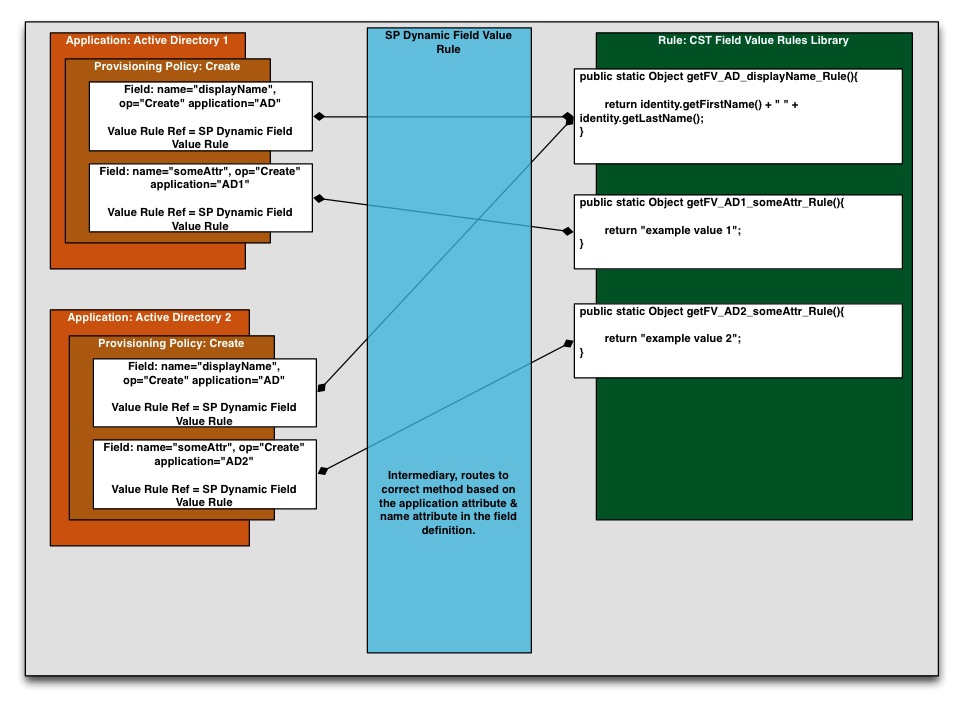


Figure - Framework - Field Value

## Dynamic Emails Framework

The Email framework will be used to simplify how emails are built and sent out during workflows. Rather than create a new step for each different type of email that a workflow needs to send out, which are usually all sent out at the end of the workflow, the workflow and underlying rules will populate a single object (a list of maps) that can then be sent into the email sub to be processed at the same time. The object model allows for extending attributes available to the templates via the object rather than via the workflow variables or step, making it easier to extend or change the emails’ content.

In any workflow, the emailArgList can be built as an ArrayList, such as:

List emailArgList = new ArrayList();

For each email, add a map object to the emailArgList. The map must contain to and emailTemplate. For example:

Map emailArgs = new HashMap();

emailArgs.put(“to”, “admin@sailpoint.com”);

emailArgs.put(“emailTemplate”, “cst Security Officer Termination Email”);

emailArgs.put(“identityName”, identityName);

emailArgs.put(“someOtherVal”, “this is a test”);

emailArgList.add(emailArgs);

Once the object is built and emails are ready to be sent, the given workflow can call the single send emails step:

<Step name=*"Send Emails"*>

<Arg name=*"emailArgList"* value=*"ref:emailArgList"*/>

<Description>

Call the standard subprocess that will handle the built-in

owner, manager and security officer approval schemes.

</Description>

<WorkflowRef>

<Reference class=*"sailpoint.object.Workflow"* name=*"SP Dynamic Send Emails Sub"*/>

</WorkflowRef>

<Transition to=*"Stop"*/>

</Step>

In the email templates, the various arguments can be referenced through the velocity dot (.) notation by simply referencing $emailArgs.<name of argument>, such as $emailArgs.someOtherVal, which would equal “this is a test” per the above example.

The following shows how the framework fits into a given LCE workflow:

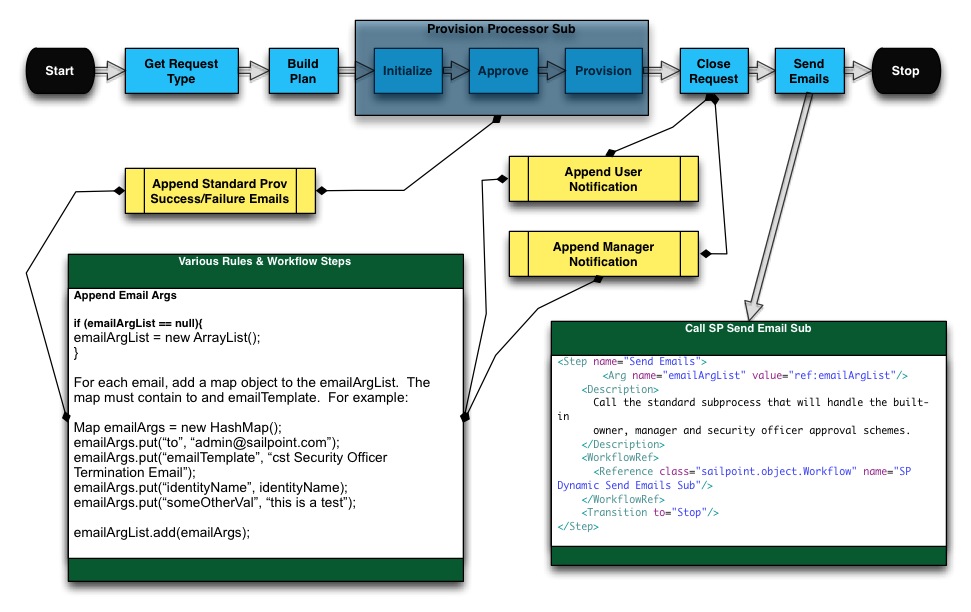


Figure - Framework - Email

## Role Assignment Framework

The Role Assignment framework can be used for dynamically building out the provisioning plan in various Lifecycle Event (LCE) workflows, namely Joiner, Mover, and Rehire workflows. It uses the native match criteria found in business roles to determine whether given roles should be added or removed from the given identity and creates the necessary account requests accordingly.

In each of the LCE workflows, there will be a step called Build Plan, which will call a specific build plan rule in a specific LCE rule library. This rule can use the role assignment framework to determine the account requests that need to be added or removed based on the current state of the identity. When used, the build plan logic is greatly simplified and looks like such:

ProvisioningPlan plan = new ProvisioningPlan();

logger.trace("Get the roles request");

**AccountRequest rolesReq** = **getAddOrRemoveRolesAccountRequest(context, identity);**

if (rolesReq != null){

logger.trace("Add the roles req");

plan.add(rolesReq);

}

The following shows how the framework fits into a given LCE workflow:

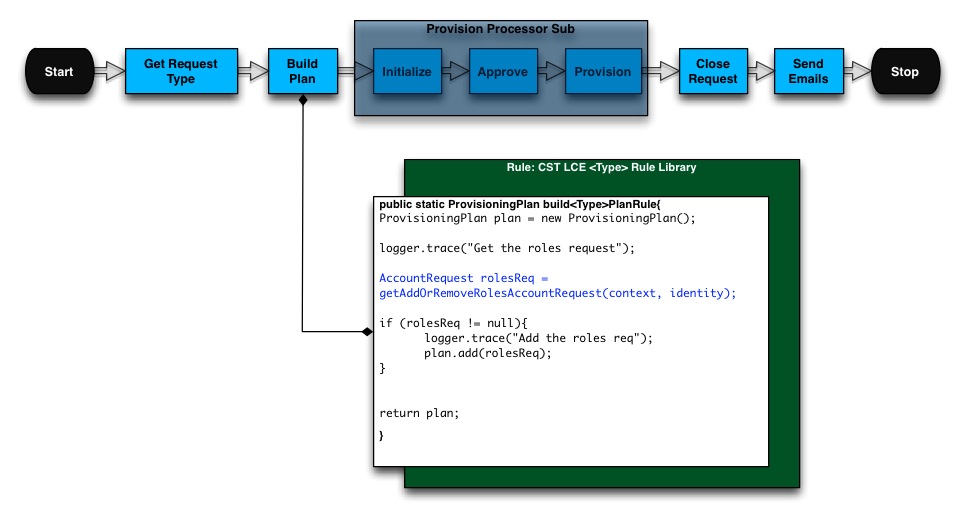


Figure – Framework - Role Assignment

## Approvals Framework

The Approval framework is a single sub process and set of configuration options that dynamically and iteratively processes all of the approval types required for a given use case, or request type. This allows for each different workflow type, or even set of requested items, to distinguish their required approval scheme and then process their unique approval requirements. For each approval type, it allows for easy configuration of the required approvers, the approval mode, and the work item config (reminders and escalation), and provides pre and post approval hooks. Just about any approval scenario can be handled and with relative ease.

The Approval Framework sub will be embedded in the Provision Processor Sub. In most cases, workflows will simply calculate a request type and call the Provision Processor Sub. Then, all approval configurations are completed in a single custom mapping object and a single rule library. The custom configuration object determines how approval types are calculated and how each given approval type behaves, predominantly by mapping steps to specific rules or methods in the rule library. The rule library contains all of the specific business logic.

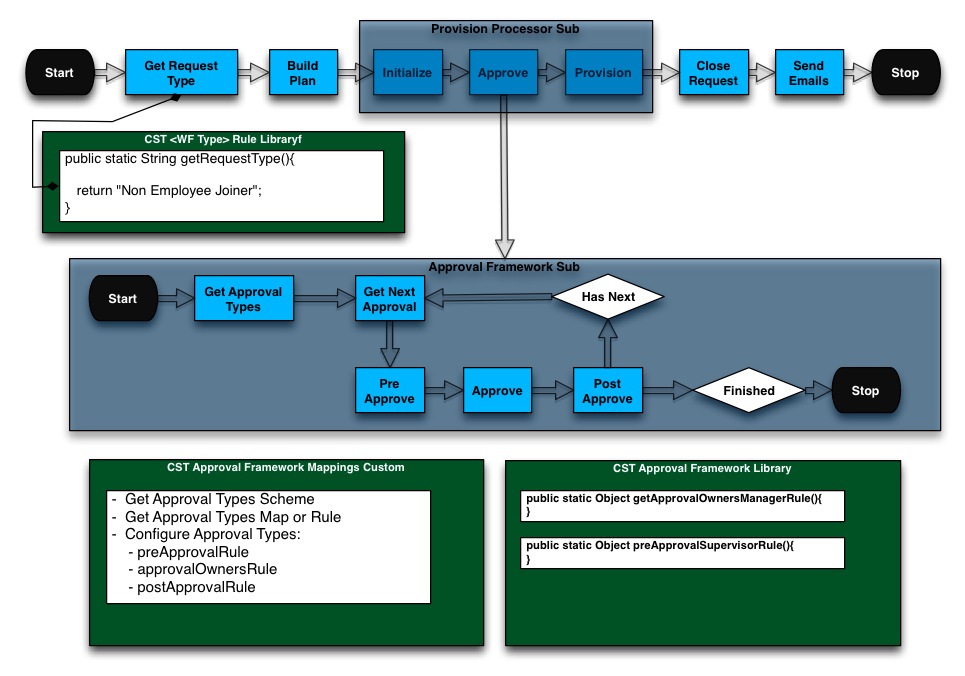


Figure - Framework – Approvals

## Provision Processor Sub Framework

The Provision Processor Sub framework is a modified version of the OOTB LCM Provisioning workflow. It encapsulates the Approval Framework sub and handles the major tasks required of just about every workflow:

* **Initialize** – Uses provisioning plan to create a provisioning project and approval set. The project is main set of instructions used for provisioning. The approval set contains all requested items that may or may not need approved. The step also creates the identity request object, which is used to track the workflow request.
* **Approve** – Calls the Approval Framework sub and processes all required approvals for the given request. This returns an updated approval set, denoting which items were approved and which were rejected.
* **Provision** – Actually provisions the given account requests via direct connection, service ticket, or work item. Returns the updated project containing a result for each request. Will handle retries.

Additionally, the sub can update the emailArgsList variable with default success/failure email arguments for each provisioning account request.

When this framework is in place, most workflows should follow a common and simplified flow:

* **Get Request Type** – Return string value denoting type of request. Useful for the underlying approval sub.
* **Build Plan** – Build the baseline provisioning request (optional if an LCM workflow and plan is built by the feature and passed in)
* **Call Provision Processor Sub** – Do all the actual work
* **Close Request** – Place for final customizations to: append additional emails, update database tables, do additional logging, etc.
* **Send Emails** – Call the Send Emails sub to process all notifications found in the emailArgsList variable.

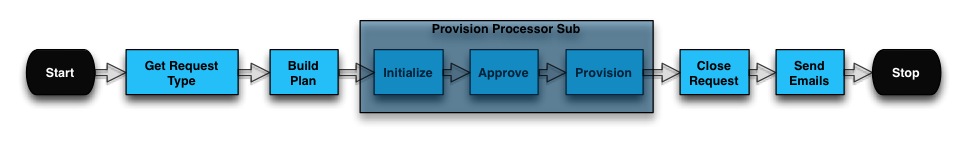


Figure - Framework - Provision Processor Sub

## Joiner Feature

The Joiner Feature is a pre-built Lifecycle Event, including the trigger object, trigger rule, workflow, underlying rule library, and a configurable mapping object and library. It is designed to simplify and streamline the development effort of most common joiner scenarios. The workflow sits atop most of the frameworks and provides a few simple configuration options.

The following options are determined in the mapping object:

1. Trigger Selector – who/when to launch the workflow
2. Birthright Assignment Type – how the initial provisioning plan is constructed
   1. Dynamic Roles – uses role assignment framework to dynamically calculate
   2. Default Assignments – uses a static list of either applications or roles
   3. Custom Rule – write custom logic to build out

Additionally, there are specific call outs or hooks in the rule library where logic can be written:

1. getRequestType – Determine request type for approvals
2. beforePlanRule – Manipulate any workflow variable before the plan is constructed
3. beforeProvisionRule – Manipulate any workflow variable after the plan is constructed
4. afterProvisionRule – Manipulate any workflow variable after the plan is provisioned

## Leaver Feature

The Leaver Feature is a pre-built Lifecycle Event, including the trigger object, trigger rule, workflow, underlying rule library, and a configurable mapping object and library. It is designed to simplify and streamline the development effort of most common leaver scenarios. The workflow sits atop most of the frameworks and provides a few simple configuration options.

The following options are determined in the mapping object:

1. Trigger Selector – who/when to launch the workflow
2. Birthright Assignment Type – how the initial provisioning plan is constructed
   1. Disable All – disables all accounts
   2. Delete All – deletes all accounts
   3. Selective Lists – uses static lists of applications to be disabled or deleted
   4. Custom Rule – write custom logic to build out

Additionally, there are specific call outs or hooks in the rule library where logic can be written:

1. getRequestType – Determine request type for approvals
2. beforePlanRule – Manipulate any workflow variable before the plan is constructed
3. beforeProvisionRule – Manipulate any workflow variable after the plan is constructed
4. afterProvisionRule – Manipulate any workflow variable after the plan is provisioned

## Mover Feature

The Mover Feature is a pre-built Lifecycle Event, including the trigger object, trigger rule, workflow, underlying rule library, and a configurable mapping object and library. It is designed to simplify and streamline the development effort of most common mover scenarios—department or manager transfer, switch from contractor to employee. The workflow sits atop most of the frameworks and provides a few simple configuration options.

The following options are determined in the mapping object:

1. Trigger Selector – who/when to launch the workflow
   1. Compare Links – evaluate attributes on given links and if changed, launch
   2. Selector – write role-style match criteria to determine whether to launch
   3. Custom Rule – write your own logic
2. Birthright Assignment Type – how the initial provisioning plan is constructed, more than one can be selected
   1. Dynamic Roles – re-evaluate the role assignment framework to add/remove roles
   2. Attribute Synch – will re-evaluate provisioning policy logic on all target accounts and provision any updates (depends on Attribute Synch Feature)
   3. Custom Rule – write custom logic to build out
3. Launch Manager Cert – whether to launch a certification

Additionally, there are specific call outs or hooks in the rule library where logic can be written:

1. getRequestType – Determine request type for approvals
2. beforePlanRule – Manipulate any workflow variable before the plan is constructed
3. beforeProvisionRule – Manipulate any workflow variable after the plan is constructed
4. afterProvisionRule – Manipulate any workflow variable after the plan is provisioned

## Attribute Synch Feature

The Attribute Synch Feature is a pre-built Lifecycle Event, including the trigger object, trigger rule, workflow, underlying rule library, and a configurable mapping object and library. It is designed to simplify and streamline the development effort of synching attribute changes from source systems to target systems, taking advantage of already built provisioning policies. The workflow sits atop most of the frameworks and provides a few simple configuration options.

The following options are determined in the mapping object:

1. Trigger Selector – who/when to launch the workflow
   1. Compare Links – evaluate attributes on given links and if changed, launch
   2. Selector – write role-style match criteria to determine whether to launch
   3. Custom Rule – write your own logic
2. Application Skip Fields – provides a map of attributes to ignore on given applications

Additionally, there are specific call outs or hooks in the rule library where logic can be written:

1. getRequestType – Determine request type for approvals
2. beforePlanRule – Manipulate any workflow variable before the plan is constructed
3. beforeProvisionRule – Manipulate any workflow variable after the plan is constructed
4. afterProvisionRule – Manipulate any workflow variable after the plan is provisioned

## Rehire Feature

The Rehire Feature is a pre-built Lifecycle Event, including the trigger object, trigger rule, workflow, underlying rule library, and a configurable mapping object and library. It is designed to simplify and streamline the development effort of bringing back a previously terminated identity, focusing on enabling current access, re-assigning birthright access, and updating target attribute values. The workflow sits atop most of the frameworks and provides a few simple configuration options.

The following options are determined in the mapping object:

1. Trigger Selector – who/when to launch the workflow
   1. Compare Links – evaluate attributes on given links and if changed, launch
   2. Selector – write role-style match criteria to determine whether to launch
   3. Custom Rule – write your own logic
2. Plan Construction Types – how the initial provisioning plan is constructed, more than one can be selected
   1. Enable Access – enable existing accounts based on a defined list
   2. Dynamic Roles – re-evaluate the role assignment framework to add/remove roles
   3. Attribute Synch – will re-evaluate provisioning policy logic on all target accounts and provision any updates (depends on Attribute Synch Feature)
   4. Custom Rule – write custom logic to build out

Additionally, there are specific call outs or hooks in the rule library where logic can be written:

1. getRequestType – Determine request type for approvals
2. beforePlanRule – Manipulate any workflow variable before the plan is constructed
3. beforeProvisionRule – Manipulate any workflow variable after the plan is constructed
4. afterProvisionRule – Manipulate any workflow variable after the plan is provisioned

# Appendix - Lifecycle Events Architecture

An Identity Refresh can be run from a number of places but is most commonly run as part of a Refresh Identity Cube task. The task can be run ad-hoc or scheduled to run for any given frequency—weekly, nightly, hourly. The task itself has many options and can be configured to run any number of times for any number of scenarios for any given population of identities. The options that are turned on dictate what is done to each identity for each refresh. Some common actions are to update identity attributes, assign or detect roles, or refresh manager status.

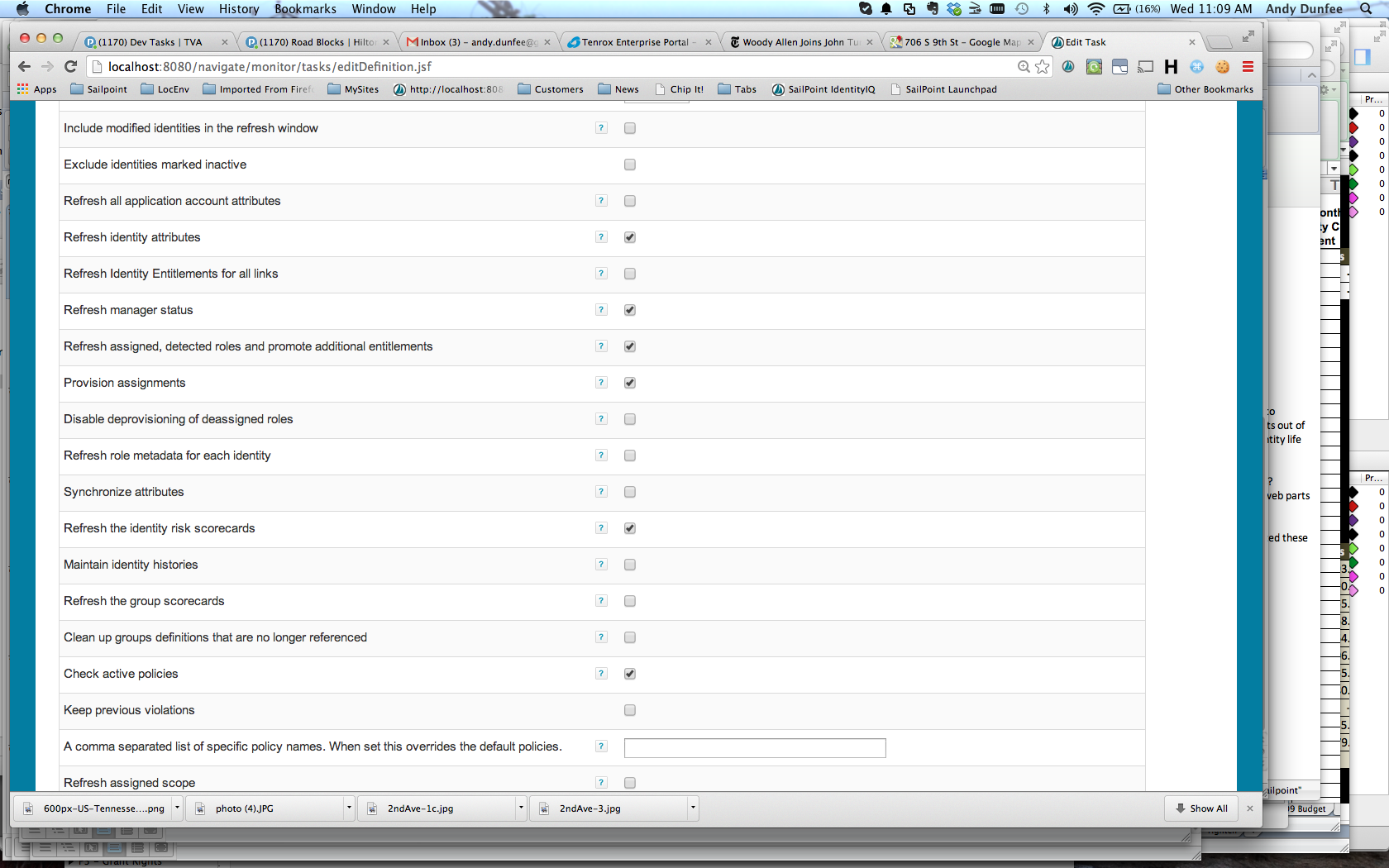


Figure - Refresh Options

The option Process Events indicates a desire to trigger Lifecycle Event workflows. If this option is on, each Lifecycle Event will be analyzed for each identity. Lifecycle Events contain two main components:

1. **The Trigger** – mechanism to determine if the workflow needs to be launched. There are different types of triggers based on the event type (Create, Manager Transfer, Attribute Change, Rule, and Native Change). For this phase of the project, the Rule type will be the most predominantly used type. The rule will have business logic to evaluate if the workflow should be launched. It will receive the previous and new identity, contain the custom logic, and return true or false. If true is returned, the workflow is launched.
2. **The Workflow** – simply put, this is the workflow that is launched and does the work for the given event.

Additionally, a Lifecycle Event has the option to be disabled and limited to certain subsets of identities based on a match list, filter, script, rule, or population.

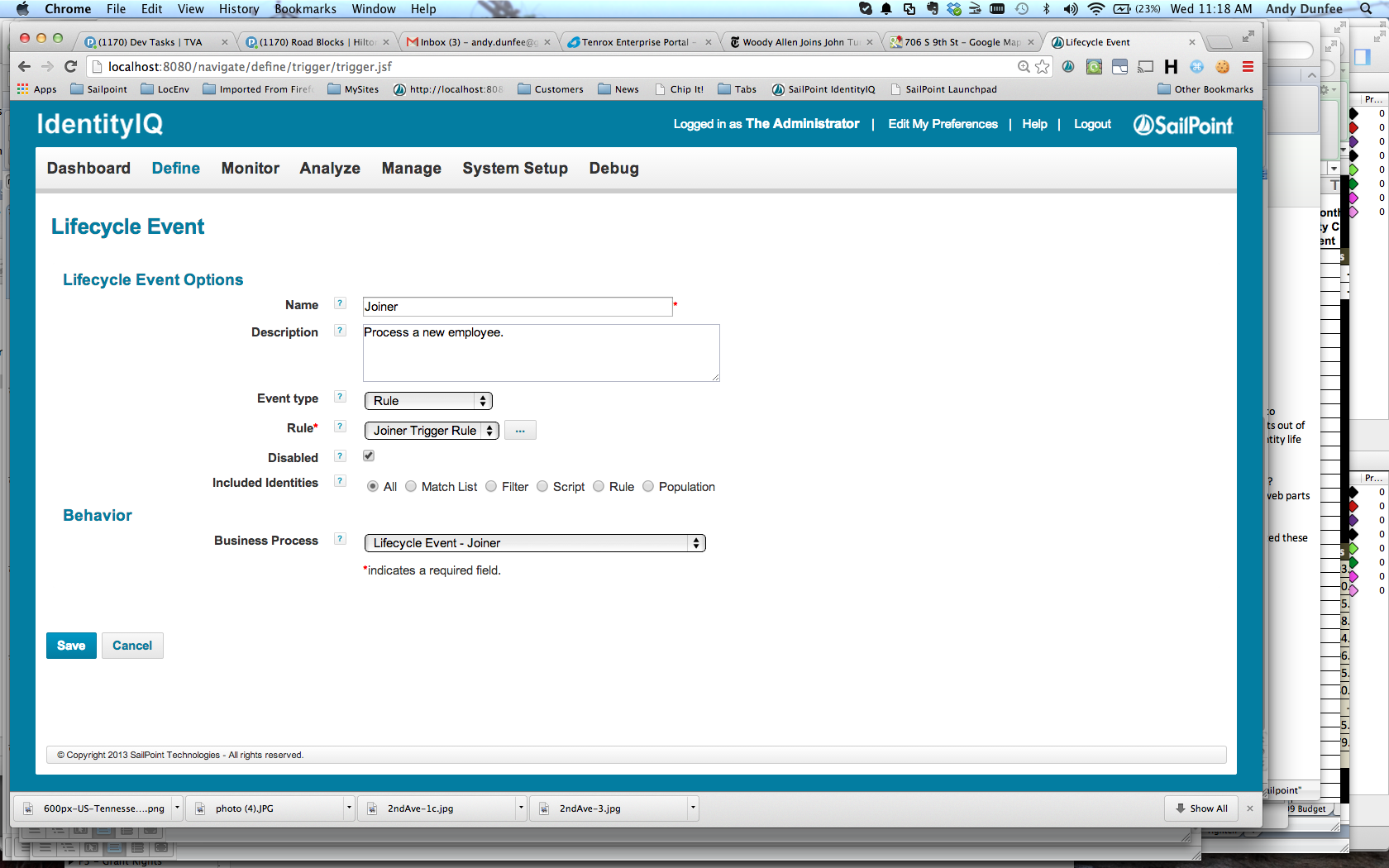


Figure - Lifecycle Event Configuration

As previously stated, all Lifecycle Events in this phase will by of type rule. The main components for each will be:

* **Lifecycle Event** – Actual container.
  + Naming Scheme: <CINIT> <NAME OF FUNCTION, e.g. Joiner> LCE
  + Options:
    - Event Type: Rule
    - Rule: Name of the rule
    - Included Identities: Additional filter logic
    - Business Process: Name of the workflow
* **Trigger Rule** – Determine if should launch. Will call method in underlying rule library.
  + Naming Scheme: <CINIT> LCE <NAME OF FUNCTION, e.g. Joiner> Trigger Rule.
* **Workflow** – Does work for event. Will sit on top of the SSF and call methods in an underlying rule library.
  + Naming Scheme: <CINIT> LCE <NAME OF FUNCTION, e.g. Joiner> WF.
* **Rule Library** – Container for all logic specific to the workflow and trigger rule
  + Naming Schema: <CINIT> LCE <NAME OF FUNCTION, e.g. Joiner> Rule Library.

All of the workflows will utilize the Provision Processor Sub, Approval, and Send Emails frameworks. They will all be tied to a specific library with common methods. In general, the flow for each will be:

* Get Request Type
* Build Plan
* Call Provision Processor Sub
* Close Request
* Send Emails

The following shows the full interaction of each event, including the trigger, the workflow, the frameworks, and the underlying rule library:

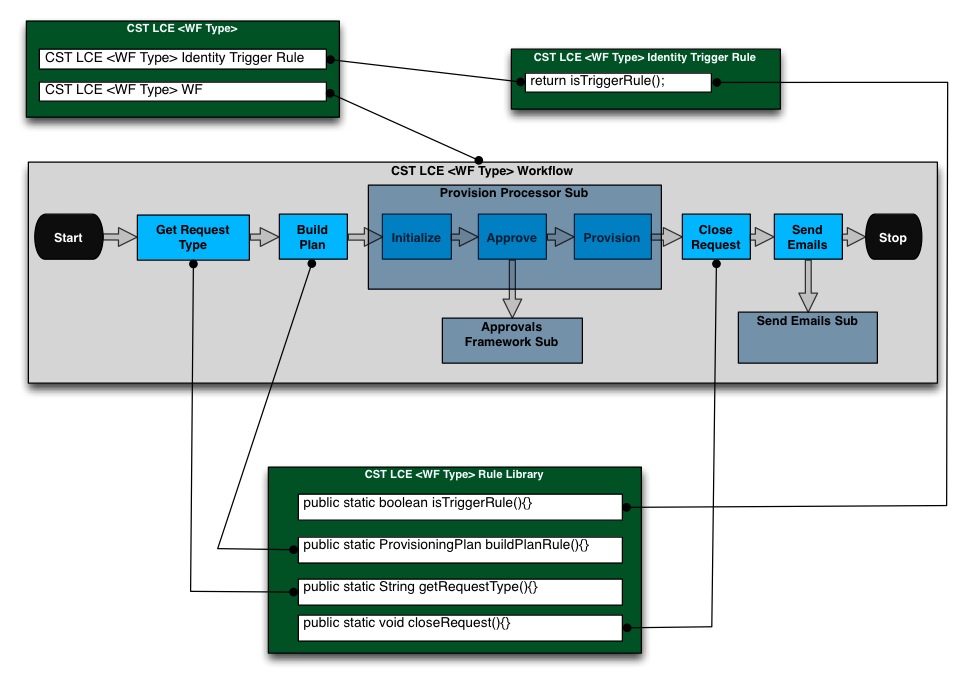


Figure - LCE Structure

Thus, with this given structure, the design of each Lifecycle Event can generally be answered with the following questions:

1. What logic triggers the workflow?
2. How is the request type determined and/or what are the required approvals?
3. How is the provisioning plan built or what does the provisioning plan contain?
4. What additional steps are required to close out the request
5. What are the email requirements?

These five questions will be answered in each Lifecycle Event with the following five corresponding sections:

1. Identity Trigger Logic
2. Request Type/Approval Logic
3. Build Plan Logic
4. Close Request Logic
5. Required Emails

Any workflow that can use a pre-built SSF Feature will amend steps 4 and 5 with before plan, before provision, and after provision hooks.