Component Test Plan

SBAC-11Program Management Interface

Version 1.6

Revision History

Date	Version	Description	Author
09/01/2013	1.0	Initial Draft	Ryan Marinello
09/18/2013	1.1	Requirement tracing	Ryan Marinello
09/24/2013	1.2	Rewriting scripts and making updates	Ryan Marinello
09/25/2013	1.3	Re working out dated scripts	Ryan Marinello
09/30/2013	1.4	Level 2 requirements reviewed and updated	Ryan Marinello
09/30/2013	1.5	Review and Summary	Ryan Marinello
10/01/2013	1.6	Edited Introduction	Ryan Marinello

Contents

I. Purpose

The purpose of this document is to describe the testing strategy and test scenarios applied to the tested component.

I. Introduction

1. The overall responsibilities of this component are:
    a. Master data repository, interface and service
    b. Supporting service for existing and future components
    c. Storage of program level information
    d. Content Management of Assets files and properties
    e. Creation and management of Asset groups
    f. Tenancy chain
    g. System property configurations
    h. Encryption and masking of configurations for security in the UI and database
    i. Maintains data and reference data needed across components
    j. Asset Group cloning
    k. History diffs per configuration line items

The user audience for Program Management is:

1) System Administrators – technical personnel responsible for configuration of properties, assets, asset groups, components and tenants.

II. Component: Program Management

The Program Management component is a shared service that acts as a supporting feature among the logical components in the assessment lifecycle. Maintaining any data and reference data that is needed across components can be accessed through the Program Management component. The user interface allows for CRUD operations for admin level users. Interaction and configuration of Tenants, Components, System Property Configurations, encrypting configurations, searching data, Assets and Asset Groups and Tenant Chain searching. Currently Test Item Bank and Monitoring and Alerting are existing services interacting with Program Management.

III. Scope of Testing

This plan focuses on manual and automated verification and validation testing of the Program Management component. Mission statement: prove level 1 and level 2 requirements for Program Management which is a supporting master repository service are functional and believed to be correct. Heuristic: CBC (consistent, beneficial, contextual)

1. Verification Testing

Verification testing will cover the functional testing of requirements.

Requirement Document	Document Name	Location	Version
Level I Requirements	DRC SBAC11 Requirements – Program ManagementLevel I	Knowledge Tree -> High Level Requirements -> DRC	V.5
Level II Requirements	DRC SBAC11 Requirements – Program Management Level II	Knowledge Tree -> Requirements -> Level II Requirements	V.2

2. Validation Testing

Validation testing will cover scenario testing.

Scenario	Script
Display and Describe the Program Management API RESTful Services and Controllers	1. http://localhost:8080/rest/api 2. /rest/api/help 3. /rest/api/assetPool 4. /rest/api/tenantchain 5. /rest/api/propertyConfig 6. /rest/api/component 7. /rest/api/tenantTypes 8. /rest/api/tenant 9. /rest/api/assetGroup 10. /rest/api/skinnableAssets

Help RESTClient
http://localhost:8080/rest/helpContent-Type: application/json
	Help Message
	Method: GET
	Response
	Content Type:
	Response Code: 200
	Response Content:
	{
	“message”: “ Help Message Here”
	}
	If an error occurs before the asynchronous request is sent, the error returns to the caller.

Create assetPool http://localhost:8080/rest/assetPool Content-Type: application/json	Create assetPool in Program Management Method: POST 1. “tenantId” 2. “name” tenantId must exist in database Response Code: 201 created { id: "5245d746e4b0d608aafbf4af" tenantId: "52448ccbe4b032d251be6dc7"

	name: "testing1" assets: null url: "/assetPool/5245d746e4b0d608aafbf4af" }

Create assetPool/{id}/assetFile
http://localhost:8080/rest/assetPool/5245d746e4b0d608aafbf4af/assetFile
Content-Type: multipart/form-data
File = browse and upload
Define file as 'assetFile' instead of 'fileUpload'
	Create assetFile in assetPool
	Method: POST
	1. "name"
	2. "type"
	3. "property"
	4. "asssetFileGridId"
	5. "assetFileName"
	6. "fileContentType"
	7. "basePath"
	8. "url"
	
	Response Code: 201 Created
	{
	"name" : null,
	"type" : "IMAGE",
	"property" : "/assetPool/assetFile/5245e493e4b0c4e284b79179/EarthHighRez.jpg",
	"assetFileGridId" : "5245e493e4b0c4e284b79179",
	"assetFileName" : "EarthHighRez.jpg",
	"fileContentType" : "image/jpeg",
	"basePath" : "http://localhost:8080/rest/",
	"url" : "http://localhost:8080/rest//assetPool/assetFile/5245e493e4b0c4e284b79179/EarthHighRez.jpg"
	}

Search /rest/assetPool Content-Type: application/json	Query String parameters Method: GET http://localhost:8080/rest/assetPool/ 1. GET /rest/assetPool/{id}
assetPool/{id} Response	{ "id": "5245d746e4b0d608aafbf4af", "tenantId": "52448ccbe4b032d251be6dc7", "name": "forclone3", "assets": [{"name": **null**, "type": "IMAGE", "property": "/assetPool/assetFile/5245e493e4b0c4e284b79179/EarthHighRez.jpg", "assetFileGridId": "5245e493e4b0c4e284b79179","assetFileName": "EarthHighRez.jpg",

	"fileContentType": "image/jpeg", "basePath": "http://localhost:8080.com/rest/","url": "http://localhost:8080.com/rest//assetPool/assetFile/5245e493e4b0c4e284b79179/EarthHighRez.jpg"}], "url": "/assetPool/5245d746e4b0d608aafbf4af" }
Remove assetPool/{id}/assetFile{id} Content-Type: application/json	Method: DELETE by URI http://localhost:8080.com/rest/assetPool/{id}/assetFile/{id} Status: 204 No Content Response does not contain any data.

Search /rest/tenantchain
Content-Type: application/json
	Query strings should return all matching data for entities
	Method: GET
	http://localhost:8080.com/rest/tenantchain?INSTITUTION=MN,Zumbro%20Education
	
	Response: Status 200 OK
	{
	searchInput:
	{
	typeAndValues:
	{
	INSTITUTION: "MN,Zumbro Education"
	}
	-
	}
	-
	tenants:
	[
	1]
	0:
	{
	id: "52448ccbe4b032d251be6dc6"
	name: "MN,Zumbro Education"
	description: "test"
	type: "INSTITUTION"
	tenantSubscriptions: null
	url: "/tenant/52448ccbe4b032d251be6dc6"
	}
	-
	-
	}

Create /rest/protpertyConfig
Content-Type: application/json
	Create propertyConfig Method: POST
	{
	  "envName" : "QA",
	  "name" : "tib_config",
	  "properties" : [{
	    "propertyKey" : "who.bar.prop",
	    "propertyValue" : "true",
	    "encrypt" : false
	  }, {
	    "propertyKey" : "pmi.nostfp.port",
	    "propertyValue" : "9999",
	    "encrypt" : false
	  }]
	}

/rest/propertyConfig response	{
	id: "524a361ce4b0da45bbe0df21"
	name: "tib_config"
	envName: "QA"
	properties:
	[
	2]
	0:
	{
	propertyKey: "who.bar.prop"
	propertyValue: "true"
	encrypt: false
	}
	-
	1:
	{
	propertyKey: "pmi.nostfp.port"
	propertyValue: "9999"
	encrypt: false
	}
	-
	-
	url: "/propertyConfig/524a361ce4b0da45bbe0df21"
	}

Search /rest/propertyConfig Content-Type: application/json	Query String parameters Method: GET http://localhost:8080/rest/propertyConfig/ 1. GET /rest/assetPool/{id}

Search /rest/propertyConfig Content-Type: application/json	Query String parameters Method: GET or by URL http://localhost:8080/rest/propertyConfig/{id}/audit Endpoint security is expected to be implemented in the future. Information currently exposed via endpoint is human readable un-encrypted data.
Search /rest/propertyConfig/name/{name}/envName/{environmentname} Content-Type: application/json	Query String parameters Method: GET or by URL http://localhost:8080 /rest/propertyConfig/name/MnA-Hyperic/envName/QA Response Code: 200
Remove /rest/propertyConfig/{id} Content-Type: application/json	Delete by endpoint Method: DELETE http://localhost:8080/rest/propertyConfig/{id} Response Code: 204

Create component by endpoint /rest/component
Content-Type: application/json
	Create a new component. Method: POST
	http://localhost:8080/rest/component
	{
	“name”:”tested”
	}

/rest/component Response 201	{
	id: "524a43abe4b0da45bbe0df23"
	name: "tested"
	url: "/component/524a43abe4b0da45bbe0df23"
	}

Update component /rest/component/{id}
Content-Type: application/json
	Method: PUT /component/{id}
	{
	“id” :524a43abe4b0da45bbe0df23”,
	“name”: “component name”
	}
	Response: 200

Search /rest/component Content-Type: application/json	Query String parameters Method: GET http://localhost:8080/rest/component/{id}/{name}

Remove /rest/component Content-Type: application/json	Component deletion Method: DELETE endpoint http://localhost:8080/rest/component/{id} Response: 204
Search /rest/tenantTypes Content-Type: application/json	Query string parameters for tenantTypes Method: GET http://localhost:8080/rest/tenantTypes ["CLIENT","STATE_GROUP","STATE","DISTRICT_GROUP","DISTRICT","INSTITUTION_GROUP","INSTITUTION"]

Create /rest/tenant
Content-Type: application/json
	Create Tenant by endpoint. Method: POST
	http://localhost:8080/rest/tenant
	{
	“name” : “Minnesota”,
	“type”:”STATE”
	}
	Response Code: 201
	{
	  "id" : "5249d53be4b0d4f99546dc19",
	  "name" : "Minnesota",
	  "description" : null,
	  "type" : "STATE",
	  "tenantSubscriptions" : null,
	  "url" : "/tenant/5249d53be4b0d4f99546dc19"
	}

Update /rest/tenant
Content-Type: application/json
	Update Tenant information by endpoint. Method: PUT
	http://localhost:8080/rest/tenant/{id}
	{
	  "id" : "5249d53be4b0d4f99546dc19",
	  "name" : "Wisconsin",
	  "type" : "STATE"
	}
	
	Response Code: 200
	{
	  "id" : "5249d53be4b0d4f99546dc16",
	  "name" : "Wisconsin",
	  "description" : null,
	  "type" : "STATE",
	  "tenantSubscriptions" : null,
	  "url" : "/tenant/5249d53be4b0d4f99546dc19"
	}

Search /rest/tenant /{id}
Content-Type: application/json
	Query String parameters for Tenant Method: GET
	http://localhost:8080/rest/tenant/{id}
	Response Code: 200
	{
	  "id" : "5249d53be4b0d4f99546dc16",
	  "name" : "Wisconsin",
	  "description" : null,
	  "type" : "STATE",
	  "tenantSubscriptions" : null,
	  "url" : "/tenant/5249d53be4b0d4f99546dc19"
	}

Search /rest/tenant/name Content-Type: application/json	Query string parameters for Tenant Name Method: GET http://localhost:8080/rest/tenant/name/Wisconsin Response Code: 200 and response content matches query above.
Remove /rest/tenant/{id} Content-Type: application/json	Deletion parameters for Tenant by ID Method: DELETE http://localhost:8080/rest/tenant/{id} Response Code: 204

Create /rest/assetGroup
Content-Type: application/json
	Create AseetGroup Method: POST
	http://localhost:8080/rest/assetGroup
	{
	  "tenant" : {
	    "id" : "5249d53be4b0d4f99546dc19"
	  },
	  "component" : {
	    "id" : "MnA0023845392489165362",
	    "name" : "testing"
	  }
	}
	
	Response Code: 201
	{
	  "id" : "5249d536e4b0d4f99546dbef",
	  "tenant" : {
	    "id" : "5249d53be4b0d4f99546dc19",
	    "name" : null,
	    "description" : null,
	    "type" : null,
	    "tenantSubscriptions" : null,
	    "url" : "/tenant/5249d53be4b0d4f99546dc19"
	  },
	  "component" : {
	    "id" : "MnA0023845392489165362",
	    "name" : "testing",
	    "url" : "/component/MnA0023845392489165362"
	  },
	  "assets" : null,

	"url" : "/assetGroup/5249d53be4b0d4f99546dc19" }

Update /rest/assetGroup
Content-Type: application/json
	Update AssetGroup Method: PUT
	http://localhost:8080/rest/assetGroup/{id}
	{
	"id" : "5249d53be4b0d4f99546dc19",
	"assets" : [{
	"name" : "updatedName",
	"assetFileName" : "test.txt",
	"type" : "FILE"
	}],
	"componentName" : "updatedComponentName",
	"tenant" : {
	"id" : "5249d537e4b0d4f99546dbf9"
	},
	"component" : {
	"id" : "MnA00971633811323258",
	"name" : "testing"
	}
	}

Search /rest/assetGroup Content-Type: application/json	Query string parameters for AssetGroup Method: GET http://localhost:8080/rest/assetGroup Response Code: 200
Remove /rest/assetGroup Content-Type: application/json	Deletion of AssetGroup Method: DELETE http://localhost:8080/rest/assetGroup Response Code 204
Search /rest/skinnableAssets Content-Type: application/json	Query string parameters for SkinnableAssets Method: GET http://localhost:8080/rest/skinnableAssets Response Code: 200
Describe the Services	Dependencies
http	UI templates
http	Asset Group Service
http	Asset Pool Service
http	Component Service
http	Navigation Service
http	Property Config Service
http	Tenant Service
http	Tenant Type Service

IV. Test Strategy

1. Test Tools

Tools may be used to:

- Invoke business rules by manipulating, creating, deleting or updating test data
- Determine test pass/fail criteria in a manual and automated fashion
- Document requirements to traceability

2. Test Environment

The DRC test environment is located at DRC and supported by the development team. The environment uses the following software, tools and frameworks:

Tool, Framework, Software, etc.	Category	Version
Java	Programming Language	1.7
Spring Framework	Framework	3.2.0
MongoDB	Database	2.0
EC2	AWS	n/a
Oracle VirtualBox	Virtual Machine	4.2.6
Apache Tomcat	Server	7.0.35
JCE	Encryption	
Hyperic	Monitoring Agent	5.0
Servlets	Platform	2.5
Advanced Rest Client Google Chrome	Testing tool	3.1.1
Hexawise	Testing tool	n/a
cURL	Testing tool	7.2.8.1
Chrome, Firefox, IE, Safari	Browsers	
FileZilla	Testing tool – SFTP	3.6.0.2

3. Test Interface

Program Management interface provides Search widget and CRUD functionality. Sorting by columns within Grids, Cloning for Asset Groups, Skinnable asset demo page and Encryption masking of configurations; In addition a REST was utilized to perform stateless API tests. Instructions for performing the tests will be included in the test scripts.

V. Test Deliverables

The table below lists the deliverables providing proof of testing.

Document Name	Type
Test Plan (this document)	PDF
Test Results (this document)	PDF
Test Scripts	PDF
Test Dashboard (this document)	PDF
Test Report (this document)	PDF

Acceptance Criteria

The Program Management component is a shared service that acts as a supporting feature among the logical components in the assessment lifecycle. Maintaining any data and reference data that is needed across components can be accessed through the Program Management component. The user interface allows for CRUD operations for users to interact with adding Tenants, searching Tenants, adding Components, searching components, adding System Property Configurations, encrypting configurations, searching configurations, adding Assets and Asset Groups. Currently Test Item Bank and Monitoring and Alerting are existing services interacting with Program Management. Consequently, the goal of testing is not to completely demonstrate that currently documented requirements have been exactly met; rather, the goal is to demonstrate that the delivered Program Management component fulfills the basic purpose of the component and can be enhanced or modified as requirements are discovered and finalized.

VI. Level 2 Requirement Revision History

Revision History

Revision Description	Author/Modifier	Version	Date
Initial working version	Russ Hammond	0.1	May 10, 2013
Version presented for internal walkthrough	Russ Hammond	0.2	May 20, 2013
Working notes, notes after initial dev meeting	Russ Hammond	0.3	May 22, 2013
Stripping out anything about roles→permissions, and other changes after talking with the team	Russ Hammond	0.4	May 23, 2013
Same as v 0.5, but removed comments for formatting purposes	Russ Hammond	0.6	July 16, 2013
Initial version of Level II placed on Knowledge Tree	Russ Hammond	1.0	July 16, 2013
Updates and revisions to LII's	Paul Krumrei	1.1	Sep 30, 2013
Updates and revisions to LII's	Ryan Marinello	1.2	Oct 02 , 2013

II. Introduction

2. The overall purpose of this component is to manage and maintain any data that is needed across the components in the system. This will include:

   a. A UI to allow storage of UI customization and branding assets
   b. A REST service to return a web-addressable path for branding assets
   c. A UI for storing system configuration data
   d. A REST service to return system configuration data

The users of Program Management will be system administrators who upload resources and set system properties. Clients of the Program Management component will be system components which request the location of a desired service, asset or property.

Terms and Definitions

Term	Definition
Asset	Data items which may include images to be displayed or included on a user interface. The data items may be associated to a component, and a tenant.
Branding	The customization of the appearance of the system for a particular Consortium client. Branding assets include logos and banners.
Configuration Management	The portion of Program Management that is responsible for externalizing properties and managing component endpoints.
Content Management	The portion of Program Management that is responsible for storing and retrieval of web assets
Endpoint	The location of a service, pointed to by a URI
Program	For the purposes of these requirements, the term “program” refers to a customer’s plan for administering tests. A program is not considered to be an entity.
URI	Uniform Resource Identifier, a string of characters used for identifying and locate a resource

Assumptions

	Assumption	Comments
1.	Users will validate the content of files they upload.	The Program Management component does not need to validate files to ensure they contain the

		correct information, are properly formatted, or have valid syntax.
2.	The system is issued with a standard set of Smarter Balanced assets. Users are responsible for creating their own assets if they wish to replace the standard asset.	~~The system is not required to provide any editing capability for assets.~~ The standard assets ~~provided with the system~~ will serve as an ~~example~~ default to be used by system administrators when customizing their configurations.
3.	The user will configure the system with all the required assets, which will be listed through documentation. If no asset is uploaded, the display of the missing asset will handled according to browser defaults.	Asset search will be coarse grained. If a tenant is configured but the asset is missing, no default asset is applied. If the tenant is not configured, base configuration assets are used.
4.	Client components will be responsible for knowing the relative URL for the endpoint of the service they wish to use. The Program Management component will provide only the base URL for the component/tenant combination.	System property configuration is a key value and the data has an option to be encrypted.
5.	~~The ability to register a specific node/server behind a load balancer is not required. It is sufficient to provide the base URL, and the load balancer will distribute requests.~~	~~System Administrators will be responsible for controlling servers/nodes behind the load balancer, and Program Management will offer no interface for that function.~~
6.	The URI's of services of one deployment may be in different domains, but they will not be in different deployment levels.	A deployment cannot use the services of another deployment, since federation is no longer required, per AIR statements in technical meetings.
6.1	Configuration set to inherit values from another configuration set.	
7.	The Program Management component will offer only one permission to CRUD the data it controls. A user granted access to set service URI's can also load content and vice versa.	There is a single audience of system administrators for the user interface of the Program Management component.
8.	If a deployment does not contain a component, there will be no configuration data for it in the Program Management component, and the user will be unable to access services of the component.	Federation is not required per statements by AIR in the technical meetings. If a user requires the services at another level of deployment, they must log in to that deployment to obtain the services.
9.	A system administrator will perform system setup and configuration. Base URL's for the endpoints for services are specified by the system administrator during system configuration.	The Program Management component does not need to provide a "registration" service that components use to register themselves.
10.	Branding assets utilized for a tenant do not vary	Per conversation with David (AIR).

	across components.	

Issues

	Issue	Status	
1.	Caching – how/when to refresh. This will only be a concern for configuration management.	Closed	5-20-13 – rdh - What support must Program Management provide for caching? Clients are responsible for keeping their cache up to date, but there may be some functionality required in PMI to support them.
2.	Roles-> Permissions – will this be in Program Management?	Closed	5-20-13 – rdh – requirements, design not yet determined
3.	Skinnable assets – which elements of the UI are to be skinnable	Closed	5-23-13 – rdh – David working with Grace, need to get a finalized list for what SBAC wants
4.	User Interface – there is not yet an approved SBAC look and feel	Closed	5-23-13 – rdh – AIR has presented a UI to SBAC but it has not been officially approved
5.	Component Inventory - is there a need for a UI to display the components?	Closed	7-17-13 – rdh – from tech meeting notes

Data and Data Relationships

The Program Management Interface will be responsible for storing and retrieving assets associated with deployments, components, and tenants. Assets may be files such as .CSS, Javascript, or properties files; Program Management may also store key-value pairs such as an encryption keys, system settings, the URI of a component, or SFTP credentials.

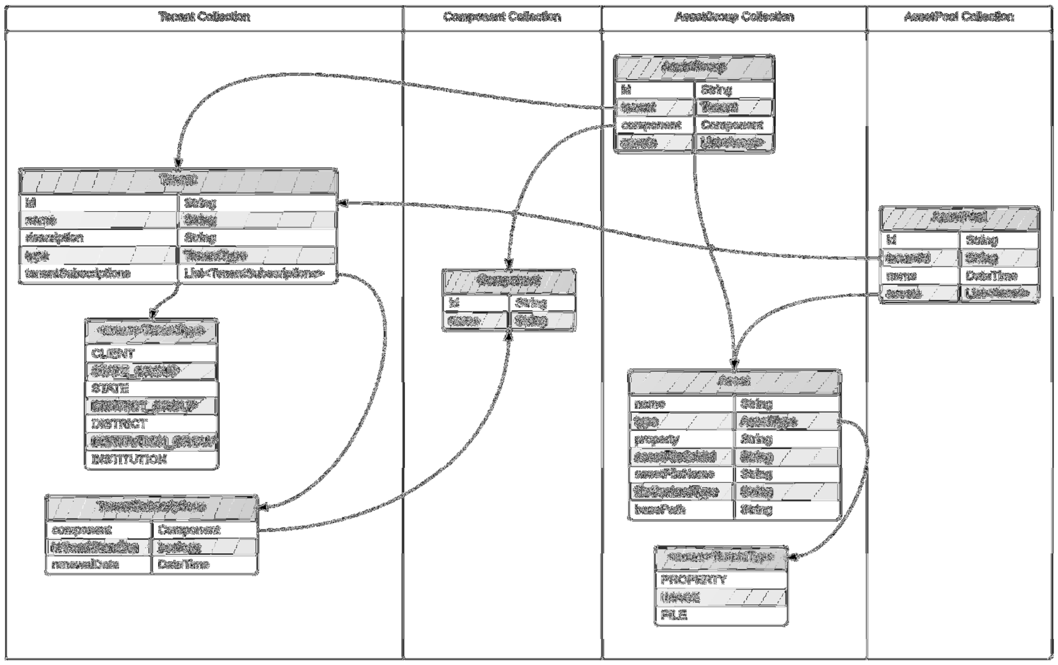

Tenant Collection
Component Collection
AssetGroup Collection
AssetPool Collection
Tenant
id
String
name
String
description
String
type
TenantType
tenantSubscriptions
List<TenantSubscriptions>
<<enum>>TenantType
CLIENT
STATE_GROUP
STATE
DISTRICT_GROUP
DISTRICT
INSTITUTION_GROUP
INSTITUTION
TenantSubscription
component
Component
renewalDate
DateTime
Component
id
String
name
String
AssetGroup
id
String
tenant
Tenant
component
Component
assets
List<Asset>
AssetPool
id
String
tenantId
String
name
DateTime
assets
List<Asset>
Asset
name
String
type
AssetType
property
String
assetFileGridId
String
assetFileName
String
fileContentType
String
iconPath
String
<<enum>>AssetType
PROPERTY
IMAGE
FILE

Image 1

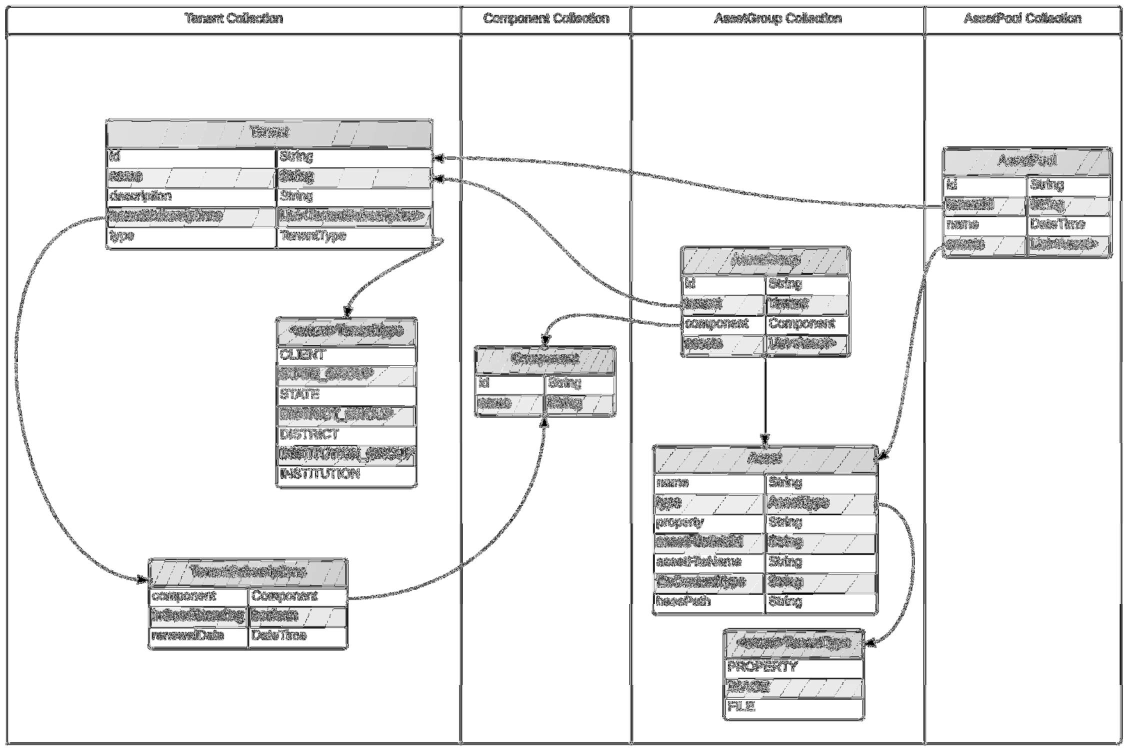
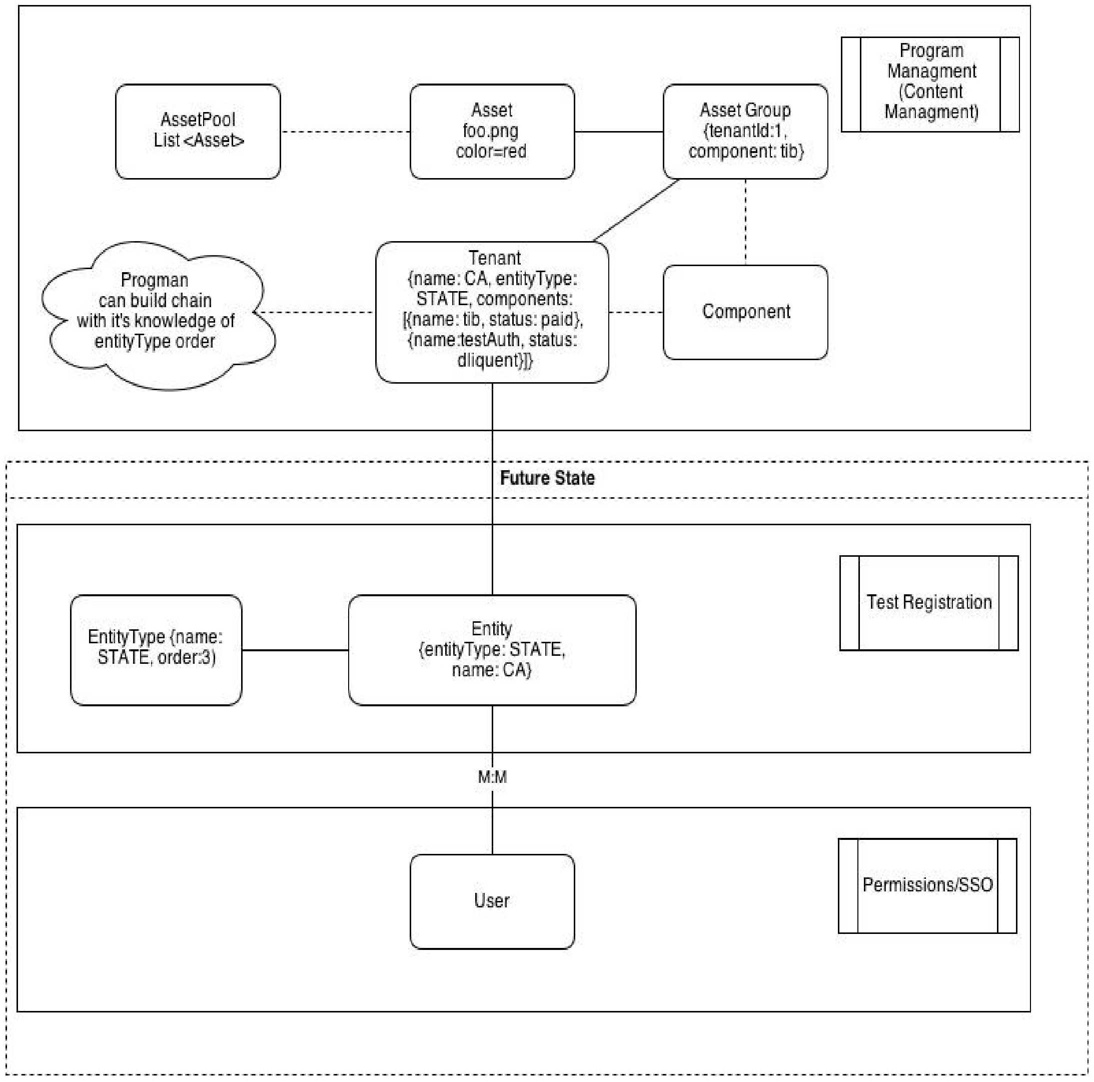


Image 2

1. Branding Assets (Skins):

The following assets are required for customizing the appearance of the user interface for Consortium tenants. The asset to be displayed is dependent on the user's tenancy. (Exact list is TBD – David is working with Grace, and needs to SBAC requirements.)

1. Logo
2. CSS file
3. Background image
4. Footer



AssetPool List <Asset>
Asset foo.png color=red
Asset Group {tenantId:1, component: tib}
Program Managment (Content Managment)
Progman can build chain with it's knowledge of entityType order
Tenant {name: CA, entityType: STATE, components: [{name: tib, status: paid}, {name:testAuth, status: dliquent}]}
Component
Future State
EntityType {name: STATE, order:3)
Entity {entityType: STATE, name: CA}
Test Registration
M:M
User
Permissions/SSO

Image 3

2. Service Endpoints

   The base URI's for services will be configured in Program Management. Client components can retrieve key-value pairs, one of which might be the URL for the component.

3. Component-Specific Assets

   Each component may have zero to many associated assets. These might include content to be included on the UI, collections of properties, or key-value pairs. See Image 1

4. System Properties

   Program Management may be required to hold additional data for system operation. Such information is key-value pairs based and may include environment settings, SFTP credentials, or. The data may be associated to a particular context such as a particular component, and may optionally be associated with a particular tenant.
   User will also have the ability to encrypt sensitive information using JCE encryption, which is U.S.

nationally supported.

Requirements

Requirements are numbered according to the following convention:

1. RFP.## - a requirement from the RFP
2. RADPM.## - a requirement from the detailed requirements from RFP-11 or the Architecture document
3. PRPM.## - a requirement from the Proposal
4. DRPM.## - a detailed requirement from the Level I document
5. OTHPM.## - other Program Management requirements

The Program Management component must meet the applicable requirements from the General Requirements.

Source.ID	Requirement	Category	Priority	Comments
RADPM.1	It is the responsibility of the Program Management [MDM] component to manage and maintain any data and reference data that is needed across components. This is to be considered as the definitive source for that information. Clarification: Program Management is assigned responsibility for storing and providing all data that defined across components. ~~At the time of this writing, this is only branding assets and component registration information.~~			5/13/13 – rdh – this can include component registration information, branding assets, and system configuration data such as public keys, the location of SFTP sites, and database properties. Met by PRPM.1, PRPM.2
RADPM.2	The Smarter Balanced architecture contains a segregation of where data is modified. The general rule is that data is only modified by the component that owns the data. For example: Item bank owns all Item related data and is the only component that can update this data. Other components consume parts of this data, but never update it. It is recommended that a simple custom solution be used instead of a commercial MDM product since the Smarter Balanced requirements do not demand a sophisticated system with features like “Single version of truth” Clarification: The Program Management component “owns” branding assets and component registration information. This information will be provided on a			

	read-only basis to other components.			
RADPM.2.1	Data added through the Program Management Component is ~~accessible~~ modifiable only through the Program Management Component.			
RADPM.2.2	The Program Management Component must provide a user interface which allows authorized users to view, add, update and delete assets and configuration data.			See PRPM.3.*
RADPM.2.3	~~The Program Management component must provide a web addressable path which will return an asset or configuration data to other components.~~			This really duplicates PRPM.2
RADPM.2.4	~~All additions, updates or deletions of Program Management controlled data must be logged~~. Program Management will maintain metadata necessary to audit changes to Program Management-controlled data.			
RADPM.2.5	~~All Program Management controlled data must have the time it was last updated, and the user who performed the update.~~			Rolled into RADPM.2.4
~~RADPM.2.5~~	~~Program Management must authenticate that the user session is valid for any service request.~~	~~General~~		Moved to OTHPM.3 since it's not related to resource ownership
~~RADPM.2.6~~	~~Program Management must check the authorizations of a user before performing any service on behalf of the user. If the user lacks the authorization, a "not authorized" error should be returned, and the attempted service request should be logged.~~	~~General~~		Moved to OTHPM.4 since it's not related to resource ownership
RADPM.3	User interfaces for all components except for Reporting must support the following browsers at the indicated version and greater: „„Internet Explorer 8+ (Windows) „„Firefox 8+ (Macintosh, Linux & Windows) „„Safari 5+ (Macintosh) „„Chrome 16+ (Macintosh, Linux & Windows)			Not in the Level I, but part of the RAD
PRPM.1	Provide the ability to store system configuration information such as style sheets, JavaScript libraries,	Data Storage		This requirement encompasses all data that may be needed across the system components. Branding

	and other system-wide information Clarification: The assets used for branding tenants will be stored by Program Management.			assets are a subset of this data.
PRPM.1.1	A listing of the default asset and configuration settings must be available to the system administrator through documentation.	Data Storage		The documentation may be external to the system, such as a System Administrator Manual.
PRPM.1.2	The system must allow a service endpoint URI to be associated to every component in the deployment. The URI stored will be the base URI for the component's services.	Data Storage		Relative endpoint information is available through the API documentation of a component. Client components must specify the URI for the specific service they are requesting, and not the general endpoint for the component. See PRPM.2.2 for URI retrieval API.
PRPM.1.3	The system must be able to associate assets to tenants.	Data Storage		This could be branding content, but possibly also SFTP sites for file uploads and downloads
PRPM.1.3.1	The system must allow ~~content~~ assets associated to a tenant to also be associated to a component	Data Storage		
PRPM.1.3.2	The system must allow ~~data~~ properties to be associated to a context.	Data Storage		This allows for the storage of key-value pairs. Properties such as parameter settings may be applicable in a given context, such as a test environment.
PRPM.1.4	The system must be able to upload files using HTTPS.	Data Storage		See PRPM.1.3 for UI reqt
~~PRPM.1.4.1~~	~~The system must be able to upload files containing UI content~~	~~Data Storage~~		Don't see a need to specify what file content can be uploaded. Files is files.
PRPM.1.5	The system must allow for the addition and storage of collections of related properties	Data Storage		
PRPM.1.6	The system must be able to support masked values.	Data Storage		
PRPM.2	Provide an interface that allows other systems to pull resources from the program management repository Clarification: RESTful interfaces will be provided by Program	API		See DRPM.5 Original AIR requirement included "in a directory structure…" However, key-value pairs meets the need to provide the assets and provides greater flexibility.

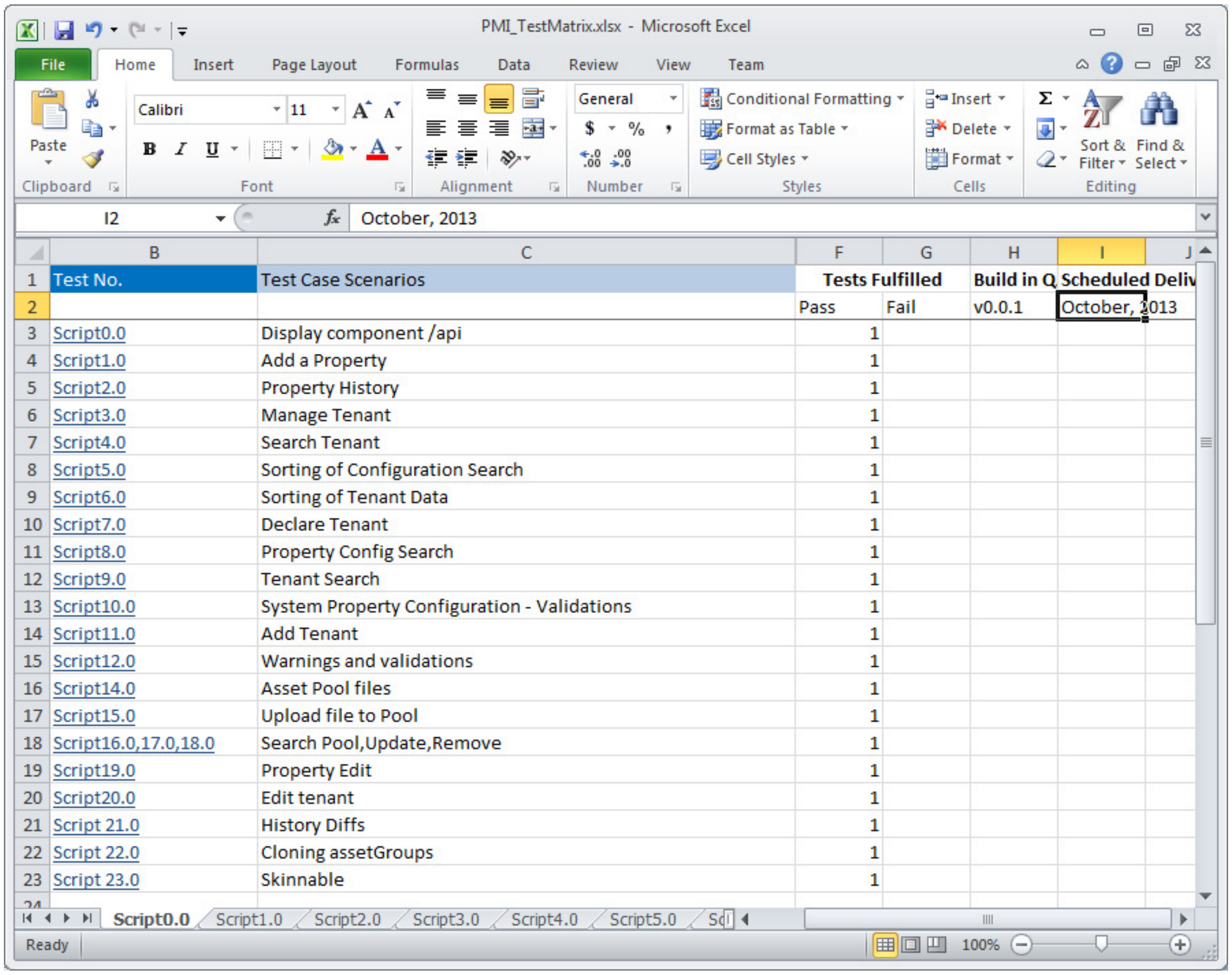
	Management to provide component registration information. Branding assets such as CSS and graphics will be provided ~~in a directory structure via a URI that components can access.~~ As a list of key-value pairs that components can access			
PRPM.2.1	Assets returned to a client component must match the tenancy of the user who initiated the request If there are existing tenant configuration matching the user tenancy but no asset matches, the service will indicate that no asset was found, and prompts you for validation of tenant and component ~~Program Management will log an error.~~	API		~~This would return a 404 to the client.. Program Management would be responsible for logging the error.~~ Tenant and Component will be defined and user promoted in the UI
PRPM.2.2	Program Management must offer a REST service that returns the list of key-value pairs for a requested component's services. If the component does not exist in the configured list of components, indicate that the component is not found and result in a displayed error.	API		This would return a 404 to the client. Program Management would be responsible for displaying the error.
PRPM.2.3	Program Management must provide a REST service which will serve properties associated to a context. If the property does not exist, indicate that the property is not found and log an error.	API		This allows for the retrieval of key-value pairs. This would return a 404 to the client.. Program Management would be responsible for displaying the error.
PRPM.2.4	If no configuration information exists for a requested tenant, assets from the base configuration may be used.			
PRPM.3	Provide a user interface that allows users of the system to associate specific content to a component and "program". Clarification: This refers to branding assets such as branding asset property or image.	UI		"Program" refers to a state's testing plans, not a type of entity. Example: The Test Authoring page might display 3 pictures. These pictures would be associated with the component, and all tenants would be expected to provide 3 pictures.
PRPM.3.1	Program Management must provide a user interface which allows files to be uploaded.	UI		This is the UI requirement
PRPM.3.2	The Program Management UI must allow assets to be associated to tenants.	UI		Branding info

PRPM.3.3	The Program Management UI must allow assets associated to tenants to be associated to components.	UI		Component-specific assets
PRPM.3.4	The Program Management UI must allow users to register key-value pairs for system components.	UI		System property configuration
PRPM.3.5	The PM UI must allow users to display stored metadata for assets in a ~~tabular~~ grid format. The display must allow filtering by deployment, component, and tenancy. The displayed table must be sortable.	UI		~~When updated, by whom, URI value, file name, etc.~~ Asset for tenant and component, Asset name, and type value preview
PRPM.3.6	The PM UI will be available only to users with the appropriate authorizations.	UI		Not in scope SSO permissions not DRC owned
~~PRPM.3.7~~	~~The PM UI must allow users to view the values of stored configurations (content assets, configuration key values)~~	~~UI~~		Covered by PRPM.3.5
PRPM.3.8	The user interface should present interfaces for adding: ● Branding assets (skinning) for a tenant ● Component-specific content ● System ~~settings~~ properties	UI		System Settings includes service registration
PRPM.3.9	Deleted			Deleted – retained numbering to keep alignment with other documents.
PRPM.3.10	The UI will not allow a null to be entered for a resource.	UI		~~Don't know of a valid use case for entering null values. If a property or asset is not needed, it should be deleted, not set to nulls.~~ Pertaining to key values, Null values are not allowed.
PRPM.3.11	The UI must prompt the user to confirm they wish to delete a resource	UI		There a dirty form and a warning message to save changes, and browser warning if you accept changes, or if you attempt to navigate from form, warning messages will be displayed
PRPM.3.12	The user interface must conform to the guidelines set forth by AIR.	UI		Following current standards set forth by AIR Standards documentation on 9/4/13
PRPM.3.13	The Program Management UI must allow properties to be associated to a context.	UI		Properties pertaining to configuration and environment.
PRPM.4	Support the SSO details obtained by logging into the portal. The program management component will	Navigation		The Program Management UI is available from the portal for

	link from the portal, retaining the credentials used at login			authorized users. See PRPM.3.x, RADPM.2.6 The portal used for the Pilot Test will be made open-source and used as the Portal for the system. SSO is not a DRC responsible component.
PRPM.5	~~Integrate with the portal component. The interface for the program management component will plug into the portal component by offering a producer interface via the Web Services for Remote Portlets Specification (WSRP) v2.0~~ Clarification: A WSRP 2.0 compliant portal is no longer required. Therefore, Program Management does not need to expose a WSRP 2.0 compliant interface.			Based on conversation with David, it is sufficient to provide a link to Program Management from the Portal. Met by PRPM.4 The portal used for the Pilot Test will be made open-source and used as the Portal for the system.
PRPM.6	Provide performance metrics, error logging, ~~and work flow alerts~~ to the monitoring and alerting component, using the framework specified previously. ~~Work flow-related alerts will flow through this component.~~ Clarification: Metrics and logging will be provided to Monitoring and Alerting as needed. However, workflow-related alerts are not required.			
PRPM.6.1	~~Any additions, updates, or deletions to Program Management controlled data must be logged. The log message should identify the data, the user making the change, and the time of the change.~~			This duplicates RADPM.2.4, RADPM.2.5
PRPM.6.2	If an error is encountered when searching, adding, updating or deleting Program Management controlled data, the error must be logged. The log entry should identify the user or client component, the service requested, the data requested, and a timestamp.			Duplicate of CPRM.6
DRPM.01	The Program Management component will store component-specific registration information including a base URI where the component RESTful interfaces can be reached.	Data Management		See PRPM.1.2
DRPM.02	An interface will be provided for registering	UI		See PRPM.3.4

	components, including information about the component including a base URI where the component RESTful interfaces can be reached.			
DRPM.03	An interface will be provided that supplies component with sufficient information to locate other components whose RESTful interfaces it seeks to interact with.	API		Duplicates PRPM.2.2
DRPM.04	The Program Management component will store branding assets such as CSS and graphics for each tenant that allow components to customize their appearance			See PRPM.1
DRPM.05	The Program Management component will store the skinnable ~~in a directory structure that is available via a URI~~ as key-value pairs.			Original AIR requirement was "in a directory structure…" but this is a "how, not a "what". Key-value pair provides the needed functionality and greater flexibility.
OTHPM.1	The Program Management will provide support needed clients to allow clients to cache data from Program Management			Relevant only to config management
OTHPM.2	Program Management will define a strategy for handling ~~data~~ properties which must be secured.			
OTHPM.3	Program Management must authenticate that the user session is valid for any service request.	General		
OTHPM.4	~~Program Management must check the authorizations of a user before performing any service on behalf of the user. If the user lacks the authorization, a "not authorized" error should be returned, and the attempted service request should be logged.~~	General		Part of PRPM.3.6. Any authenticated user can request properties/assets from Program Management

VII. Test Dashboard



I2 October, 2013

	B	C	F	G	H	I
1	Test No.	Test Case Scenarios	Tests Fulfilled		Build in Q	Scheduled Deliv
2			Pass	Fail	v0.0.1	October, 2013
3	Script0.0	Display component /api	1			
4	Script1.0	Add a Property	1			
5	Script2.0	Property History	1			
6	Script3.0	Manage Tenant	1			
7	Script4.0	Search Tenant	1			
8	Script5.0	Sorting of Configuration Search	1			
9	Script6.0	Sorting of Tenant Data	1			
10	Script7.0	Declare Tenant	1			
11	Script8.0	Property Config Search	1			
12	Script9.0	Tenant Search	1			
13	Script10.0	System Property Configuration - Validations	1			
14	Script11.0	Add Tenant	1			
15	Script12.0	Warnings and validations	1			
16	Script14.0	Asset Pool files	1			
17	Script15.0	Upload file to Pool	1			
18	Script16.0,17.0,18.0	Search Pool,Update,Remove	1			
19	Script19.0	Property Edit	1			
20	Script20.0	Edit tenant	1			
21	Script 21.0	History Diffs	1			
22	Script 22.0	Cloning assetGroups	1			
23	Script 23.0	Skinnable	1			

VIII. Executive Test Summary

Our primary test objective was to prove Program Management component was capable to execute level I and level II requirements received for required functional areas which are covered in the test plan.

Our focus was testing component interactions and component deployment through URI, REST operations, RESTful web API HTTP controllers, layers, JSON responses, database collections, JCE encryption salts, UI/persistence, database and services .

Testing coverage consisted of positive validation for creation of tenants, components, system property configuration files, asset uploads, asset groups, component subscriptions, encryptions, inheritance of configurations, cloning of asset groups, search widget functionality, filtering, sorting, tenancy chain with has asset group or with has tenancy. Additional options were tested for Collection URI/Element URI's which are read and modified by CRUD (Create, Read, Update, and Delete) operations.

Program Management tests were performed for positive scenarios as well as negative scenarios targeting user interface validations against data objects, controllers and services, and cascaded flat file related objects. Database tests targeted connection, encryption, read, write, removes and audits as well as performance of endpoint inputs, database injection and UI. Browser performance and compatibility testing UI was performed cross browsers using Chrome, Fire Fox, Safari and IE where default settings were tested. http: tests covered look and feel standards, services, functionality, logic and usability.

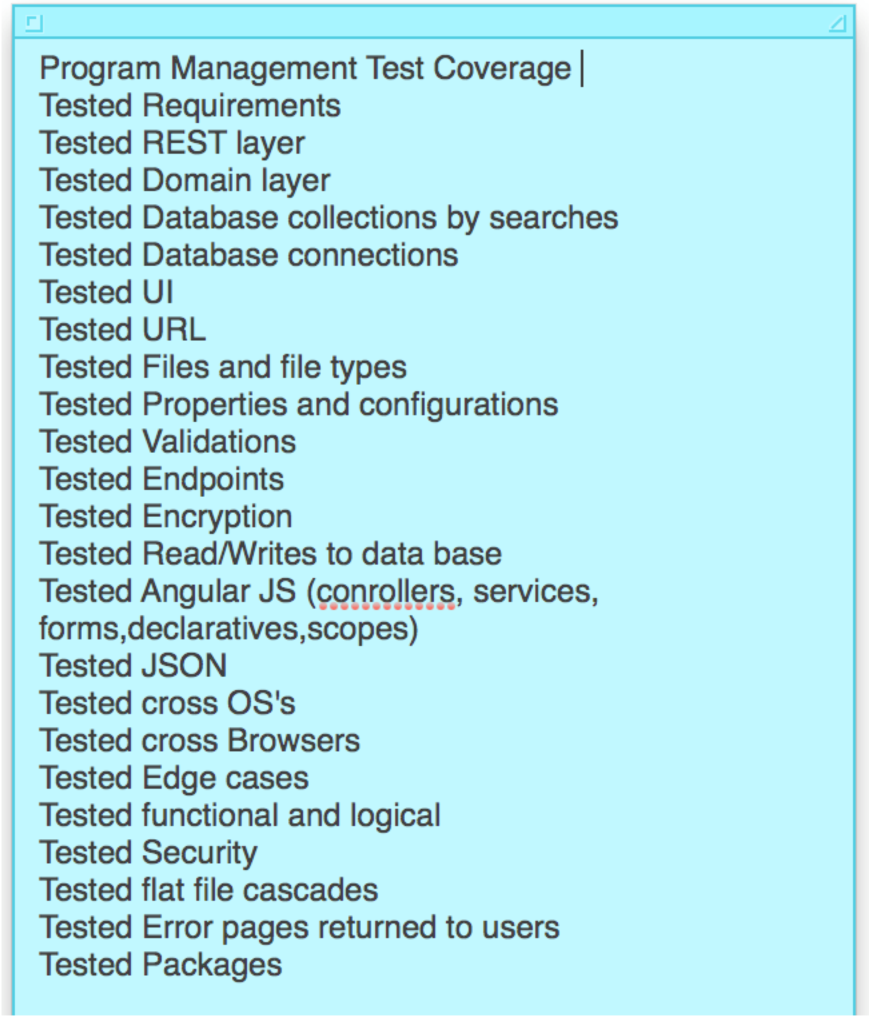
UI testing was based from supporting UI AIR guidelines. Unique icons for each functional area, uniform pages and consistent layouts as well as cohesive design make up the Program Management component.

Test Equipment / Tools – intel core i5 laptops accessing the /api and using either Fire Fox, IE,Safari or Chrome browsers and their respective REST client plugins. Mac book pro with retina quad core i7 and a linux Ubuntu 12.04

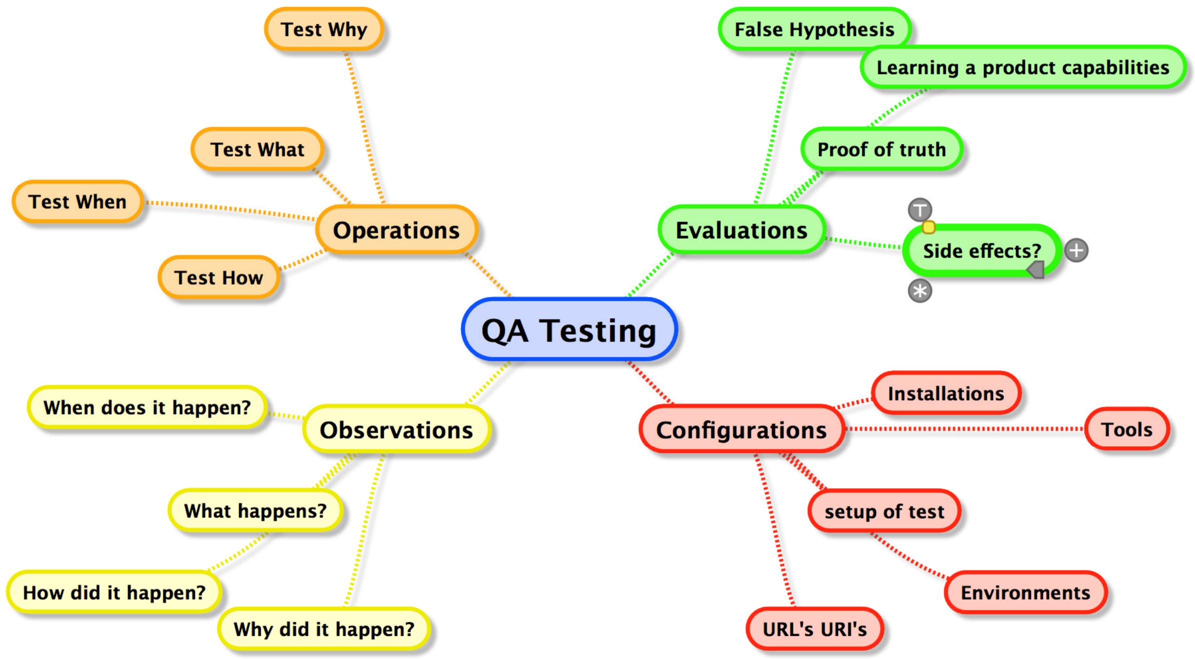
All defect discoveries or improvements revealed during testing were quickly reported, managed and fixed in JIRA. Test coverage analysis compiles from combinatorial pairs, cobertura reports and requirements.

The heuristic test strategy, a simple model was used considering the requirements, tests, and environment and product elements.

Overall testing of the Program Management component shows good design, reliability, scalability, usability, and capability as a supporting features component among other shared services and for expected future component dependencies.



Program Management Test Coverage
Tested Requirements
Tested REST layer
Tested Domain layer
Tested Database collections by searches
Tested Database connections
Tested UI
Tested URL
Tested Files and file types
Tested Properties and configurations
Tested Validations
Tested Endpoints
Tested Encryption
Tested Read/Writes to data base
Tested Angular JS (conrollers, services, forms,declaratives,scopes)
Tested JSON
Tested cross OS's
Tested cross Browsers
Tested Edge cases
Tested functional and logical
Tested Security
Tested flat file cascades
Tested Error pages returned to users
Tested Packages



QA Testing
Operations
Test Why
Test What
Test When
Test How
Evaluations
False Hypothesis
Learning a product capabilities
Proof of truth
Side effects?
Observations
When does it happen?
What happens?
How did it happen?
Why did it happen?
Configurations
Installations
Tools
setup of test
Environments
URL's URI's