

**FOS Fleet Management Design**

April 4, 18

Author: *Todd Clark, Stanley Wang*

**Document Control Information**

**Document Information**

|  |  |
| --- | --- |
| **Name** | Freestyle Fleet Management Design |
| **Program Name** | Freestyle |
| **Author** | Todd Clark and Stanley Wang |
| **Status** | Draft |

**Document Edit History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | Date | Description | Author |
| 7.0 | 11/20/2017 | Initial Version | Todd Clark |
| 7.1 | 12/7/2017 | Updates after conversations with, BSquare, Candid, Sales force, and CMS teams | Todd Clark |
| 7.2 | 12/21/2017 | Clarified ER Diagrams based on Xian team feedback.  Ruleset is a separate object.  Warning message on simulate or deploy  One validation response format to sales force A5 and A14  Added TO DO in A8 for Stanley  Updated A2 to comply with DMS-172, and DMS-185  A12 should trigger clean up of old artifacts  Add Artifact Type to A12  Cover A4 with no Target State | Todd Clark |
| 7.3 | 1/10/2018 | Changes related to sales force and CMS integration | Todd Clark |
| 7.4 | 2/8/2018 | Merging changes of FOT design resulted from meeting on 2/7/2018. | Todd Clark and Stanley Wang |

# Overall problems to solve.

The purpose of this document to is to define how all of the Freestyle Ecosystem components will work together to enable device management. At the time of this authoring coke is engaged with two venders BSquare and Candid (using AWS) to build a prototype device management system. This document will define how the new device management system provided by either vender **must** interact with Freestyle. There are 9 core problems that must be solved to implement device management for Freestyle. The document describe how we will use the saga pattern to solve.

#P1 Calculate the target state of artifacts that should be deployed to a dispenser from rules (most of the rules come from CMS).

#P2 Calculate the content of and Job for a dispenser. A Job is an atomic update for a dispenser (an atomic update could mean content bundle install, content bundle removal, and/or software update)

#P3 Enrollment/Un-Enrollment

#P4 Synchronize settings between the dispenser and FET, report on settings.

#P5 Executing a Job and tracking the status of a job.

#P6 Reporting debugging of #5 above.

#P7 Installation of Job on a dispenser (means content bundle install, content bundle removal, and/or software update)

#P8 Remote control of the dispenser.

#P9 Other weird corner cases related to the first 8 problems

This document will focus on platform interaction in the context of solving each of these problems.

The APIs defined in the document at labeled A1, A2, A3 … successively and these labels match the sequence and other diagrams.

Pay special attention to APIs A4 and A7 these are used repeatedly to solve multiple problems.

# Calculate the target state of artifacts that should be deployed to a dispenser from rules (most of the rules come from CMS).

**Begin** of Summary to FOT design as outcome of 2/7/2018 meeting

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

The FOT team is strongly urged not to deviate from the design guidance wherever practical. Any item numbered with BR. Should be considered compulsory, they can’t be met this need to be noted and the doc must be changed to something that can be met.

1. **Table Schemas**

The object/table schema will be an extension of the FET schema as show in Fig 1 and Fig2 below.

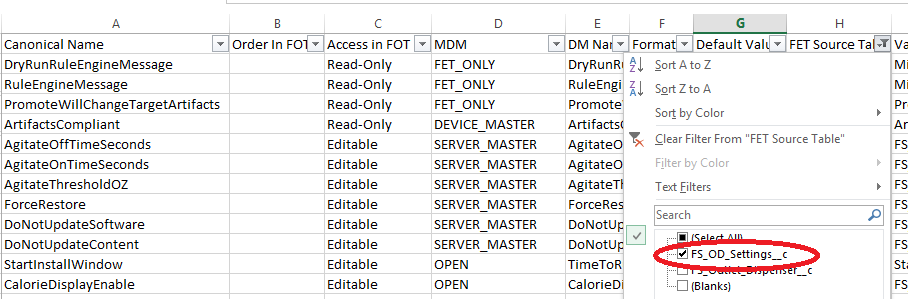
**Areas of concern where existing prototype schema may or no be aligned are highlighted in red.**



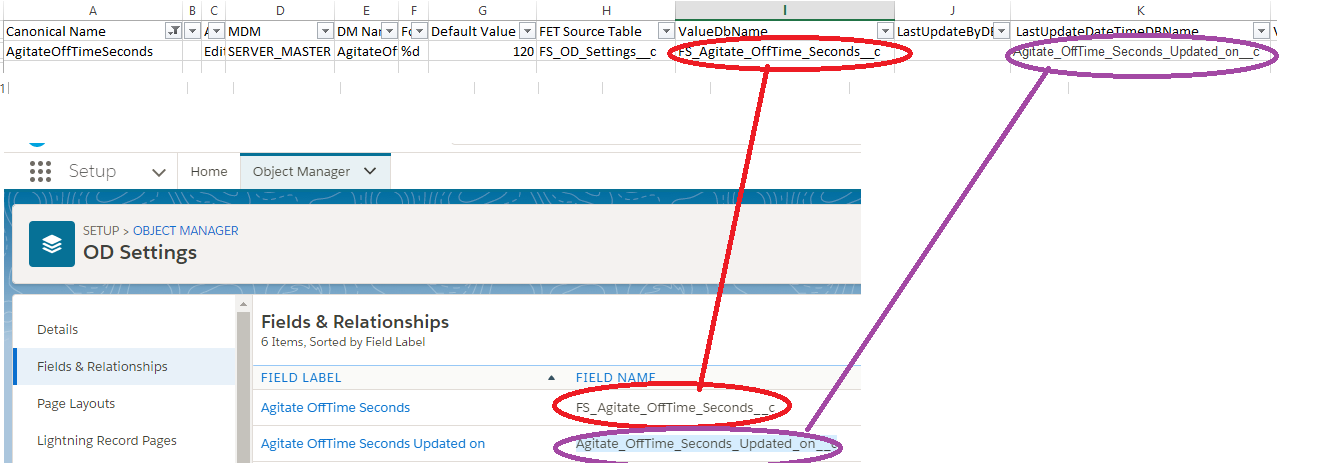
Fig 1

**BR.FOT.S.1** All field highlighted in red will be added to the OD Record table. Note the settingVerison entry is purely for FET and FOT use the integration layer does not use this.

**BR.FOT.S.2** For each setting in SettingTemplates.csv where FET Source Table = FS\_OD\_Settings\_\_c and MDM is not set to OPEN the will be two fields in the setting table, one field for the value and one for the last update date.

****

The kindergarten matchup between settingtemplates.csv and FOT

****

**BR.FOT.S.3** For each setting in SettingTemplates.csv where FET Source Table =

FS\_OD\_Settings\_\_c and MDM is set to OPEN the will be 4 fields in the setting table

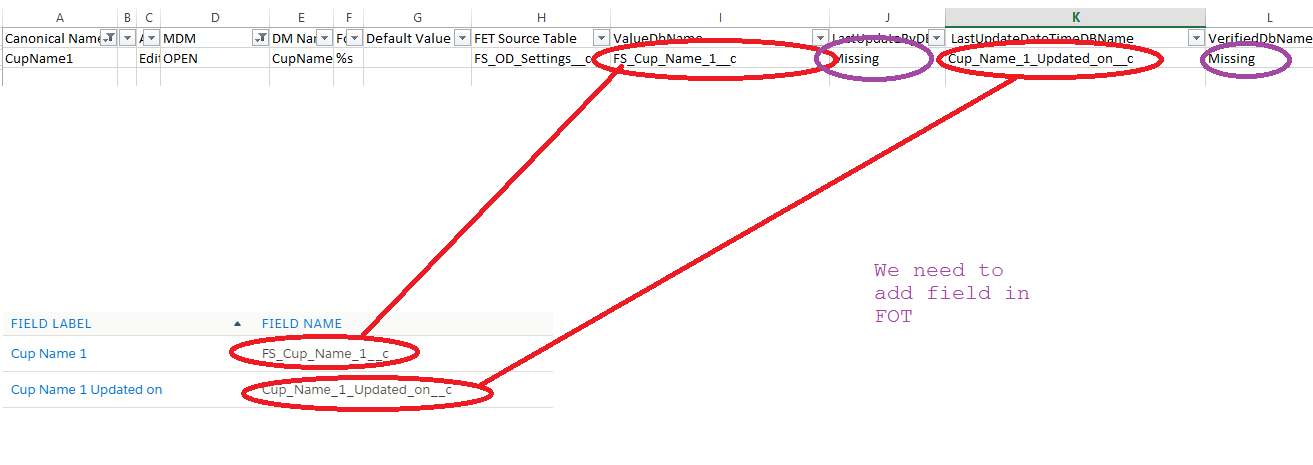
For each OPEN or FIRM\_ONLY setting, setting table must contain 4 values:

1. A value to hold the value of the setting
2. LastUpdateBy a pick list set to (FOT, RulesEngine, Device)
3. LastUpdateDateTime (FOT will set the LastUpdateDateTime when the field is changed in the UI. FIRM will on set on value change)
4. Verified

FOT will set to false on value change.

FIRM will set to true when the value change is verified from the dispenser.

The kindergarten match up.



The kindergarten matchup between settingtemplates.csv and FOT

**BR.FOT.S.4** Whenever there is an OD record created the Salesforce system must automatically create and setting table record with the record type set to normal. There must be one and only one setting table record for each OD record with a record type or normal.

**BR.FOT.S.5** Whenever creating a setting table record (with record Type = “normal” ) use the default values from SettingTemplates.csv.

**BR.FOT.S.6** All values in all table shown in Fig 1 must be available for ad hoc reporting.



Fig2

**BR.FOT.S.8** Artifacts and RuleSet tables will be provided as shown above

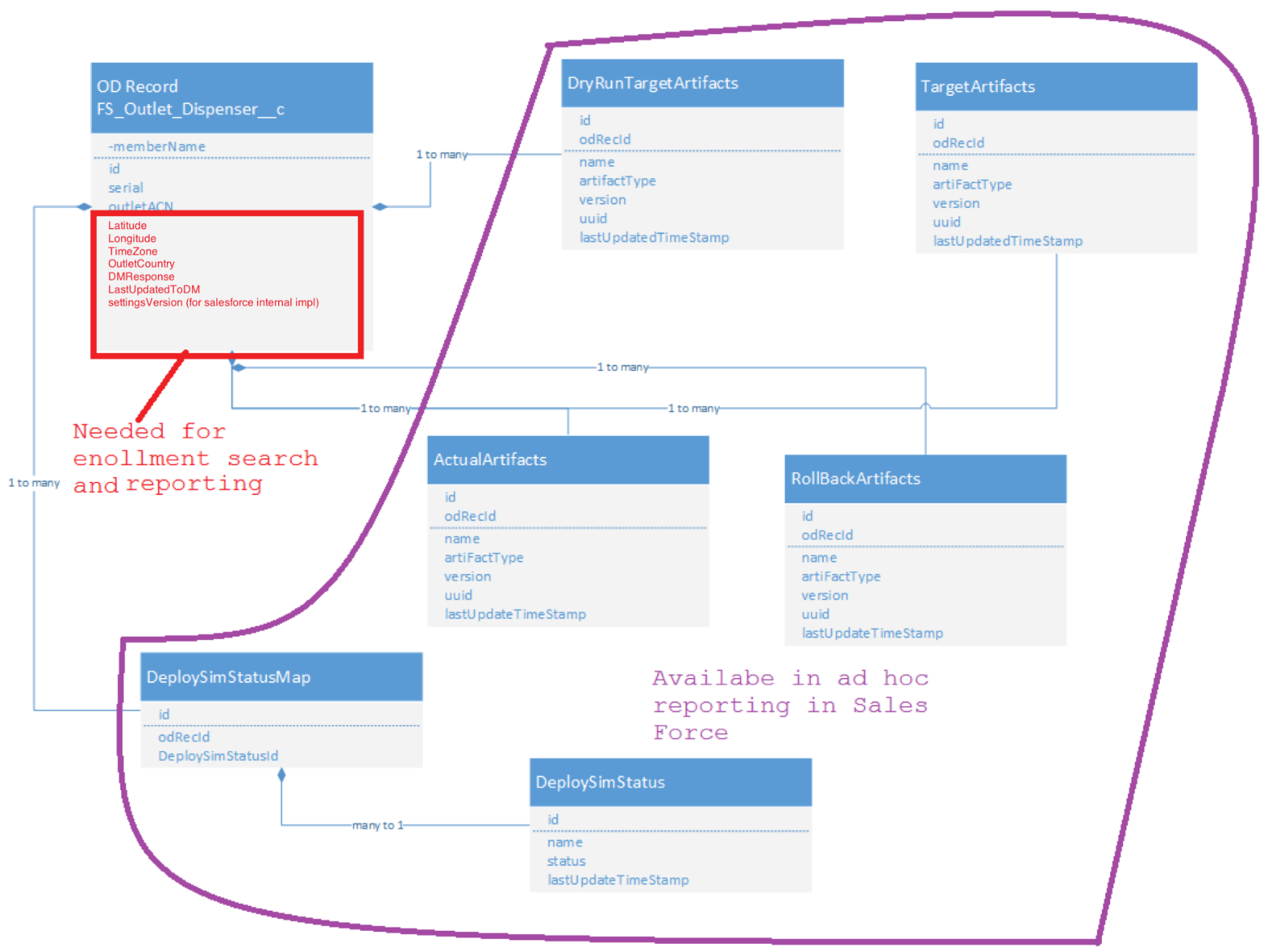
**BR.FOT.S.9** Simulation Summary Row Table will be provided for reporting. This table is populated by FIRM when A3 is called.

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

**End** of Summary to FOT design as outcome of 2/7/2018 meeting.

The below ER models have been refactored to match above Summary.

The entity model in the Salesforce org that currently houses FET will **logically be updated** to include the below.



TO DO Stanley add dryRunRuleEngineMessage message to the above

Fig P1.1.1

The entity model in the Salesforce org that currently houses FET will be **physically be updated** to include the below.

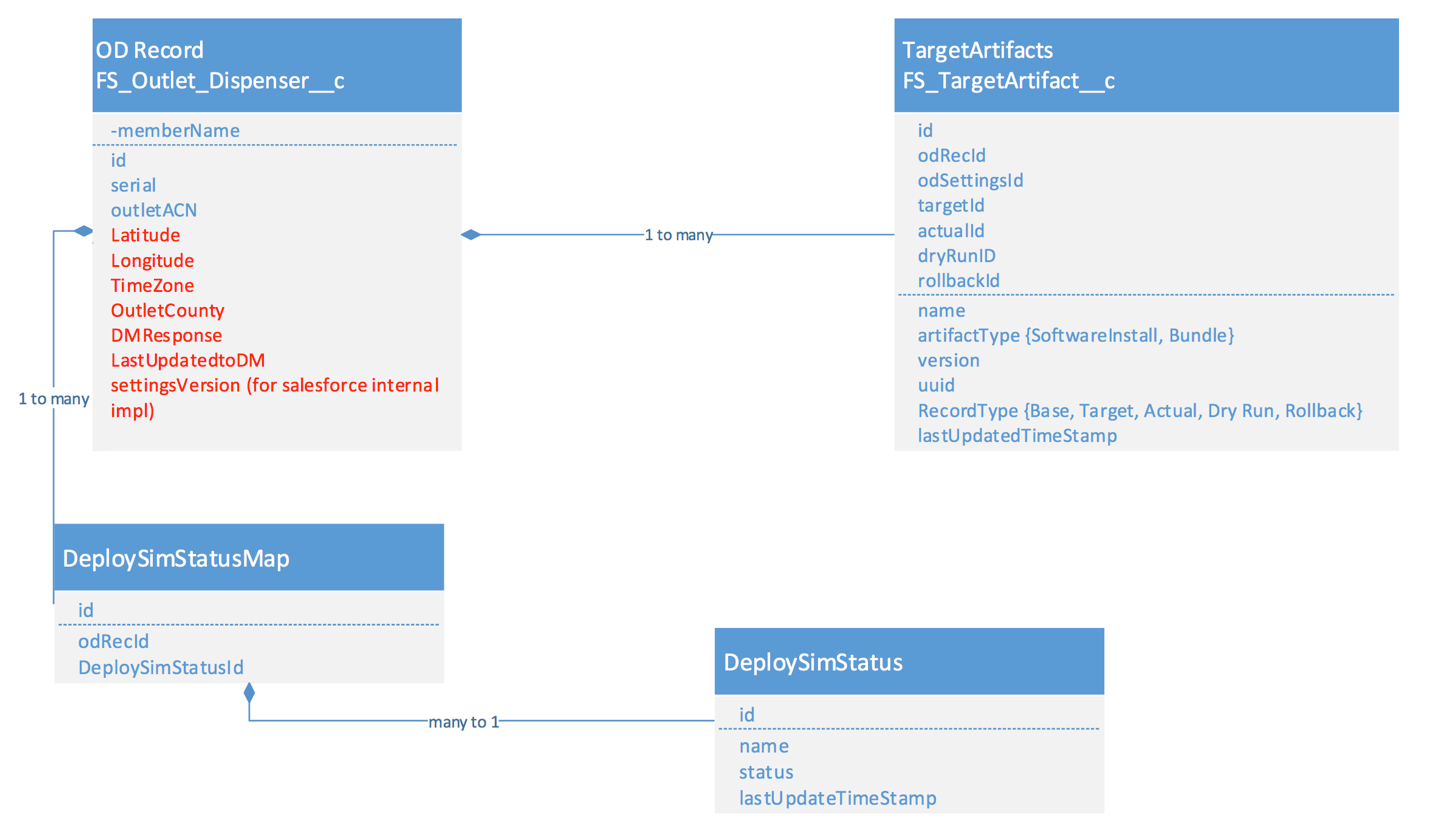


Fig P1.1.1

RecordType in “TargetArtifact” table will be set one of DryRunTargetArtifacts, TargetArtifacts, ActualArtifacts, and RollBackArtifacts.

From here forward in this document the logical tables will be referenced.

The OD record is an existing table other tables are new tables in Salesforce. The natural primary keys for the OD record are serial and outlet ACN. Latitude, longitude, timeZone, and addressHash will be added to the OD record to enable searching for enrollment APIs near an address.

DryRunTargetArtifacts, TargetArtifacts, ActualArtifacts, and RollBackArtifacts table will be populated by the integrations layer and available for reporting.

The DeploySimStatus table will be loaded with two rows by default. There only be 2 rows in this table.



This table is used to track the status for simulation and deployment jobs below.

And there will be two entries in DeploySimStatusMap for every OD record.

Additionally, three logical tables will be added.

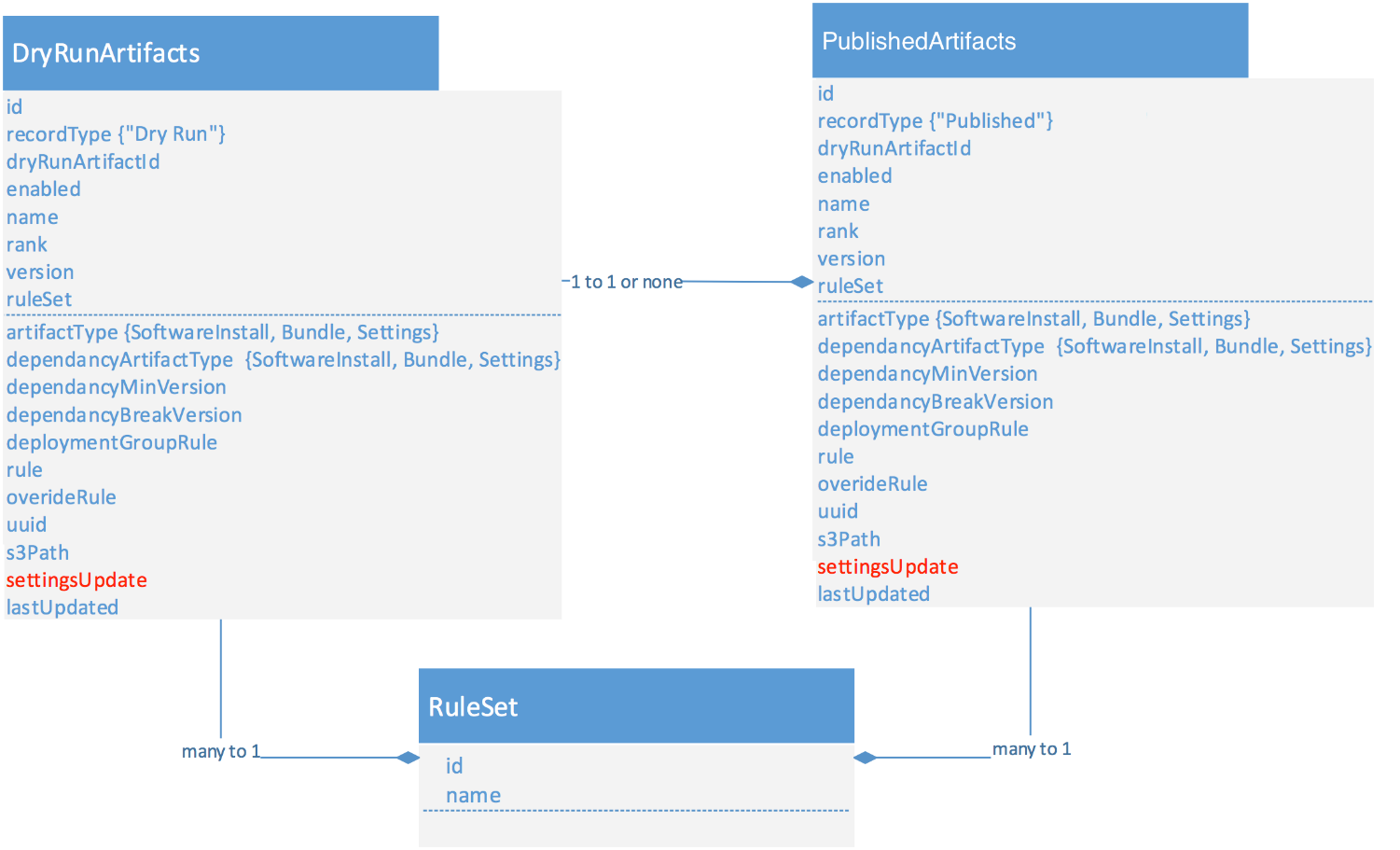


Fig P1.2

Physically this will be 2 tables in Salesforce. Record Type differentiates dry run vs not dry run (“Published”).

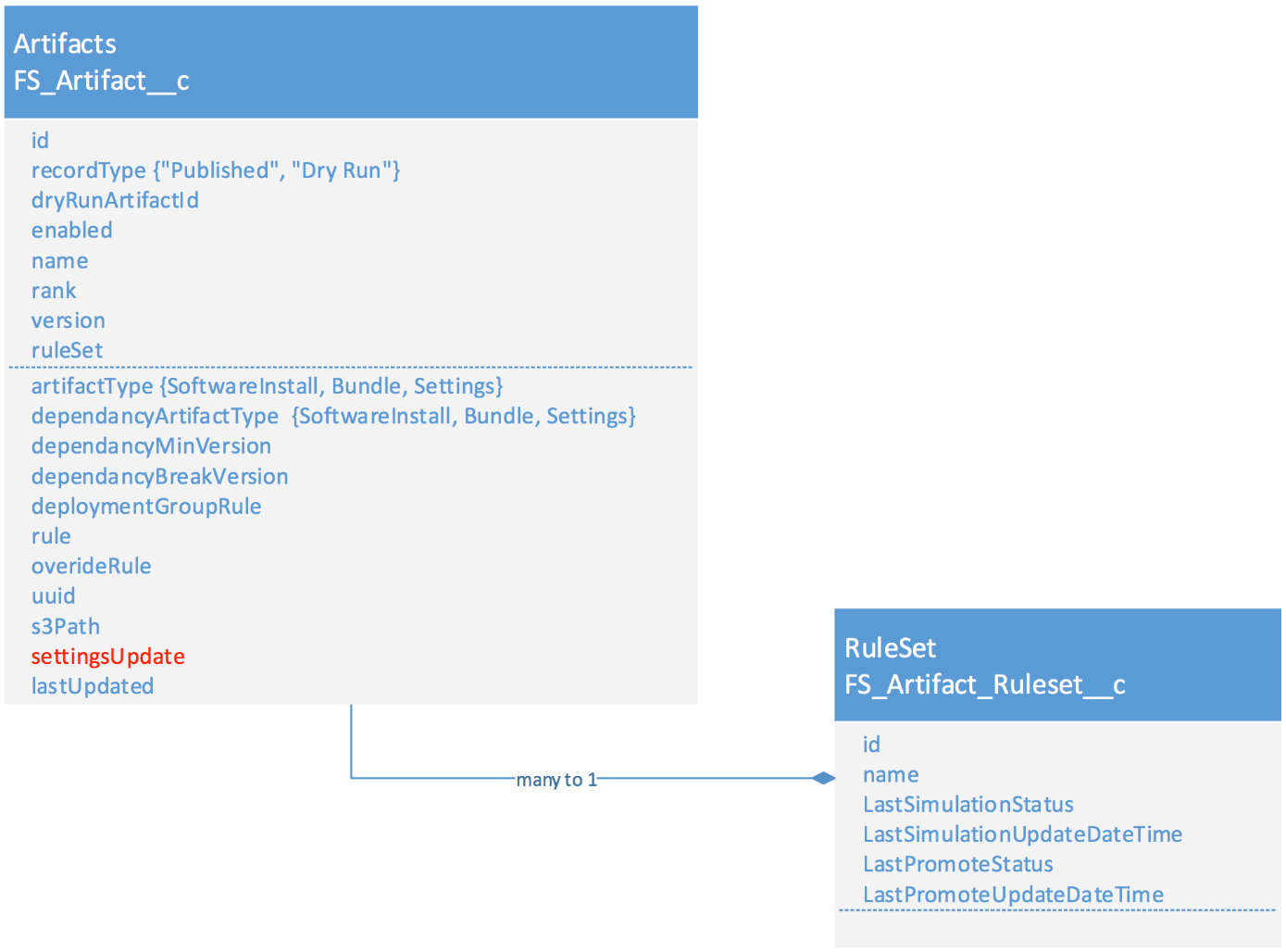


Fig P1.2.1

Going forward the logical rather than physical representation will be referenced.

The ArtifactFacts data is used to calculate the target state of a dispenser. That is to populate the TargetArtifacts Table.

DryRunArtifacts is used populate the DryRunTargetArtifactFacts table for simulation purposed before promotions before promotion.

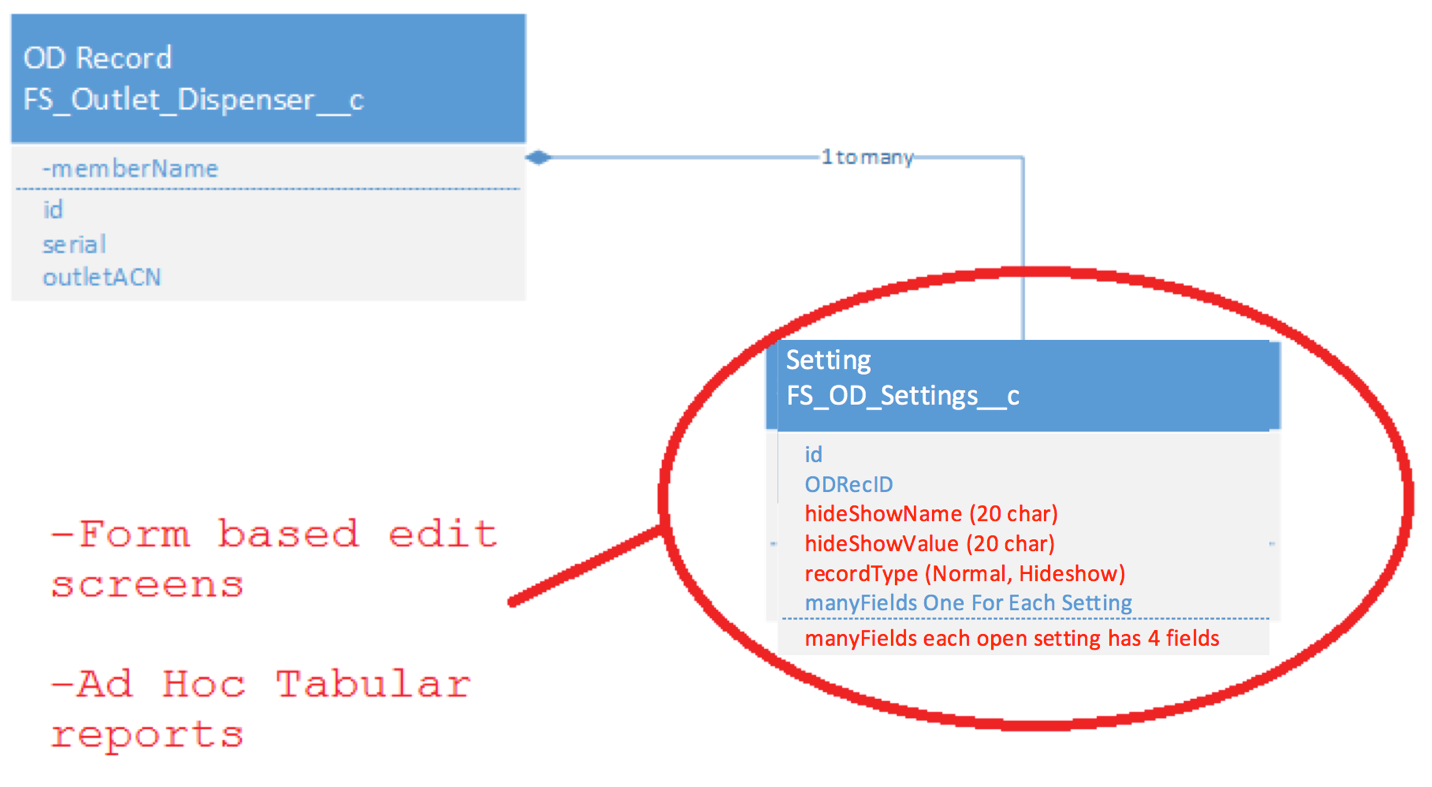


Fig P1.3

The diagram below describes the overall process for publishing rules and calculating the target state.

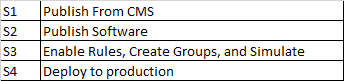


Fig P1.4

**Begin** of Summary to Salesforce UI Requirements as outcome of 2/7/2018 meeting

1. The UI Navigation

The overall navigation should have the sections shown below

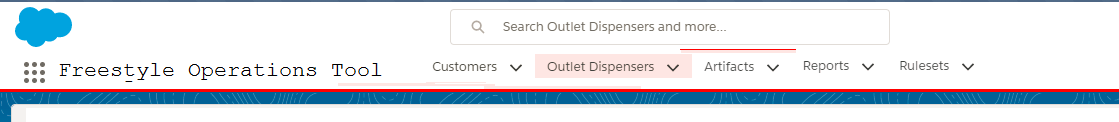


Fig 3

1. The Artifacts Tab

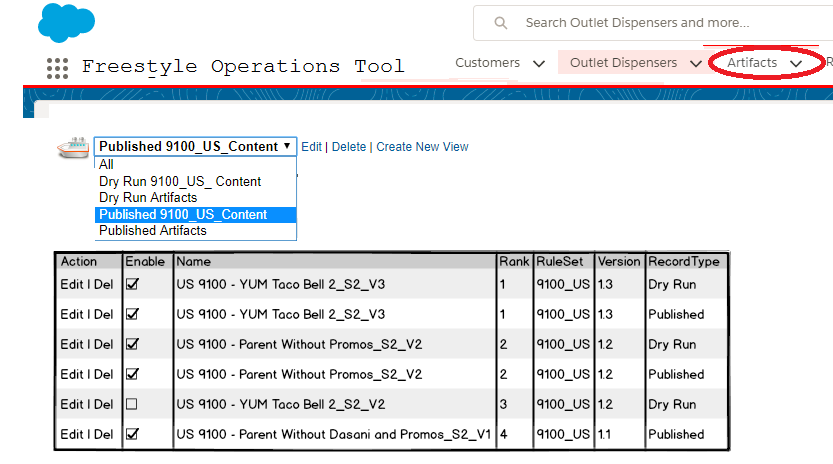


Fig 4

**BR.FOT.UI.A.1** Clicking on Artifacts takes the user to a screen similar to the above the default views for “All”, “Dry Run Artifacts”, and “Published Artifacts” should show the fields above in fig 4.

**BR.FOT.UI.A.2** Operations management should be capable of creating views for each ruleset as shown above and sharing across the team.

Once the user selects an artifact the user will navigate to a screen like fig 5 below.

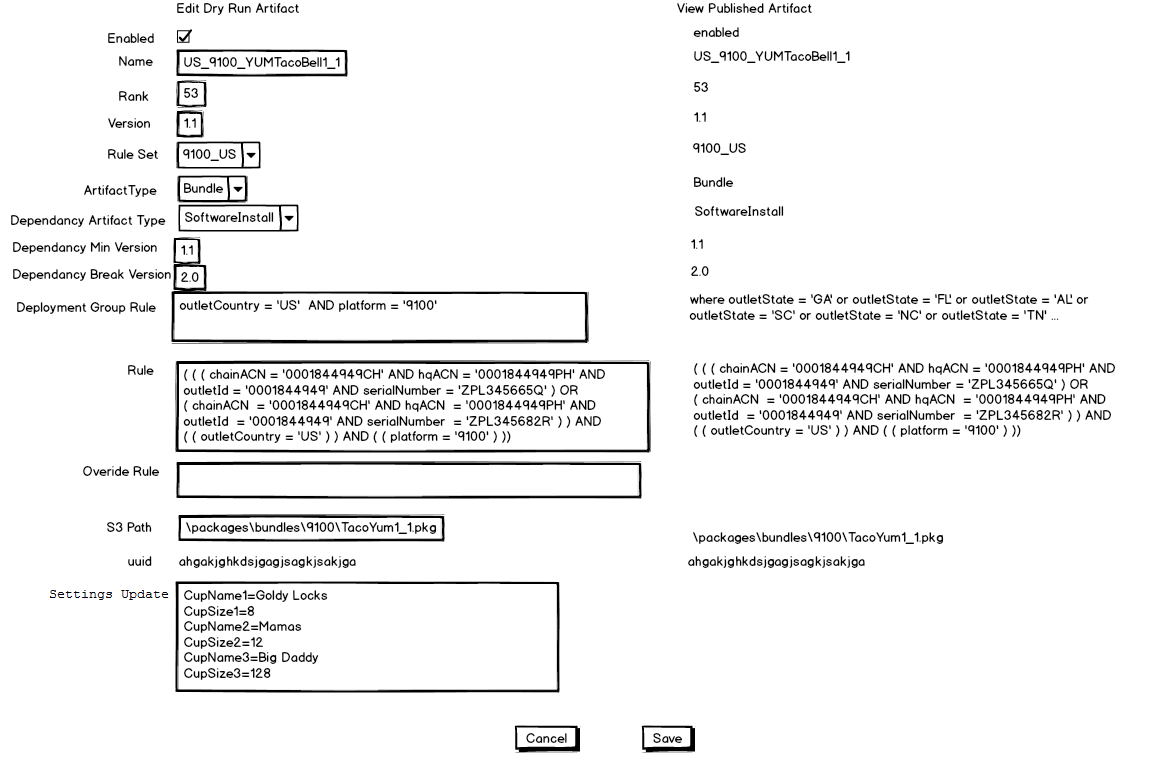
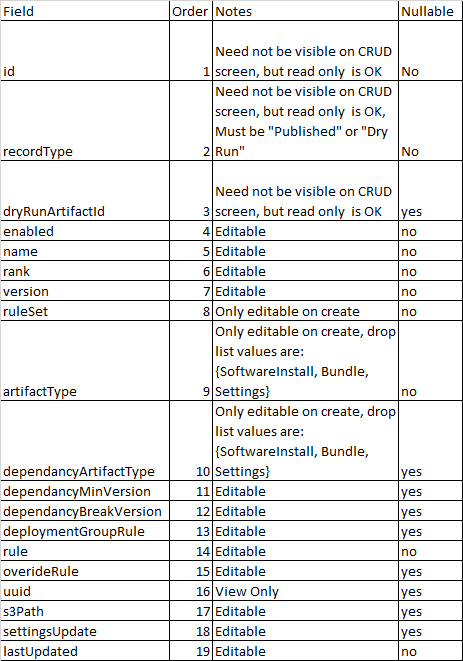


Fig 5. Above is an example where operations is widening the scope of an artifact by changing the “Deployment Group Rule”. After operations save the Dry Run Artifact they will promote this ruleset. This will copy the change for “Deployment Group Rule” to the published artifact.

**BR.FOT.UI.A.3** Fields will appear in the order shown in Fig 5. The published artifact will appear alongside the “Dry Run”. If it exists data will appear this was “Dry Run” on the left and “Published” on the right without regard for if the published or dry run artifact is selected from Fig4.

**BR.FOT.UI.A.4** The Published Artifact is never editable.

**BR.FOT.UI.A.5** The table directly below applies the behavior of each field.



**BR.FOT.UI.A.6** A Published Artifact can only be deleted if both the Published and “Dry Run” artifact are disabled.

**BR.FOT.UI.A.7** A Dry Run Artifact can only be deleted if there is no Published Artifact, and the Dry Run Artifact is disabled.

**BR.FOT.UI.A.8** If any artifact is deleted the routine below will be used to fix the rank and remove and gaps. This list of artifacts in the below is a list of all artifacts in the ruleset after the delete.

**private** **void** fixRank(List<Artifact> artifacts) {

//TO DO Sort artifacts from lowest numerical

//To highest numerical rank that is 1 is first

**int** rank = 1;

**for** (Iterator iterator = artifacts.iterator(); iterator.hasNext();) {

Artifact artifact = (Artifact) iterator.next();

artifact.setRank(rank);

rank++;

}

}

**BR.FOT.UI.A.9** If an artifact is updated with the same rank as another do this to bump other artifacts in the ruleset and align the ranks.

**private** **void** bumpAndFixIfNeeded(Artifact artifactToSave){

List<Artifact> artifactsMatchingRankAndRuleSet = **null**;

//TO DO select a list from the DB

//matching artifactToSave.getRuleSet() artifactToSave.getRank() and

//where artifactToSave.getId() does not match

**if** (artifactsMatchingRankAndRuleSet.size() > 0){

List<Artifact> artifactsWithGreateRankeInRuleSet = **null**;

//TO DO select a list from the DB

//matching artifactToSave.getRuleSet() and rank > artifactToSave.getRank() and

//where artifactToSave.getId() does not match the id

**for** (Iterator iterator = artifactsWithGreateRankeInRuleSet.iterator(); iterator.hasNext();) {

Artifact artifact = (Artifact) iterator.next();

artifact.setRank(artifact.getRank() + 1);

}

//TO DO save artifactsWithGreateRankeInRuleSet to the DB

List<Artifact> artifactsInRuleSet = **null**;

//TO DO select a list from the DB

//matching artifactToSave.getRuleSet()

**this**.fixRank(artifactsInRuleSet);

//TO DO save artifactsInRuleSet to the DB

}

}

**BR.FOT.UI.A.10** If the user change the s3 path FOT will set UUID to null.

**BR.FOT.UI.A.11** If the artifact type is set to SoftwareInstall or Bundle then the S3 path is visible and Setting Update field is not visible.

**BR.FOT.UI.A.12** If the artifact type is set to Settings then the S3 path is not visible and Setting Update field is visible.

**BR.FOT.UI.A.13** When the user clicks save FOT will call A5 to validate the data on screen.

If FOT gets a response indicating valid is true (like below) if will display the message provided and the text for yes and not provided. If the user choses yes FOT will save the data, otherwise FOT will not save the data.

{

"valid": "true",

"hashCode": "",

"message": "Your rule and group applies to 5032 dispensers without regard for other artifacts in ruleset",

"textArea":"",

"noMsg":"No",

"yesMsg"; "Yes"

}

If FOT gets a response indicating valid is not true (like below) FOT will display the message provided and FOT will not save the data.

{

"valid": "false",

"hashCode": "",

"message": "Failed with sql exception blah blah blah,

"textArea":"",

"noMsg":"If you see this in FOT it is a bug",

"yesMsg"; "If you see this in FOT it is a bug"

}

\*Note there are a large number of validation requirements in Jira for creating and editing artifacts calling A5 handles all of them for you in FOT. NO WORK IN FOT. JUST CALL A5.

1. The RuleSets tab

Recall that Figure 5 is an example where operations is widening the scope of an artifact by changing the “Deployment Group Rule” in the Dry Run Artifact. To move the Dry Run Artifact to the published states the user will need to select the ruleset and simulate and then promote the ruleset.

Selecting the rules set to do this depicted in Fig 6 below.

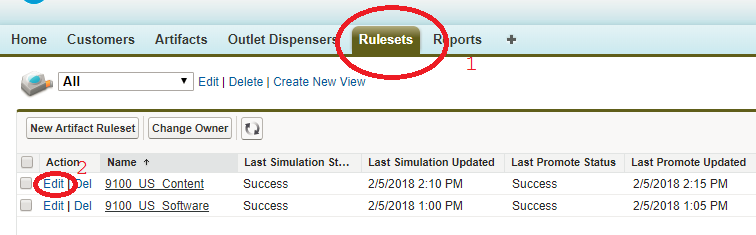


Fig 6 User clicks to select the ruleset

**BR.FOT.UI.R.1** By default there is an all view users can define new views for the ruleset.

Once the user clicks to edit the ruleset they see the screen below. Users can:

* Simulate the ruleset
* Promote the ruleset
* Or edit artifacts in the ruleset from this screen

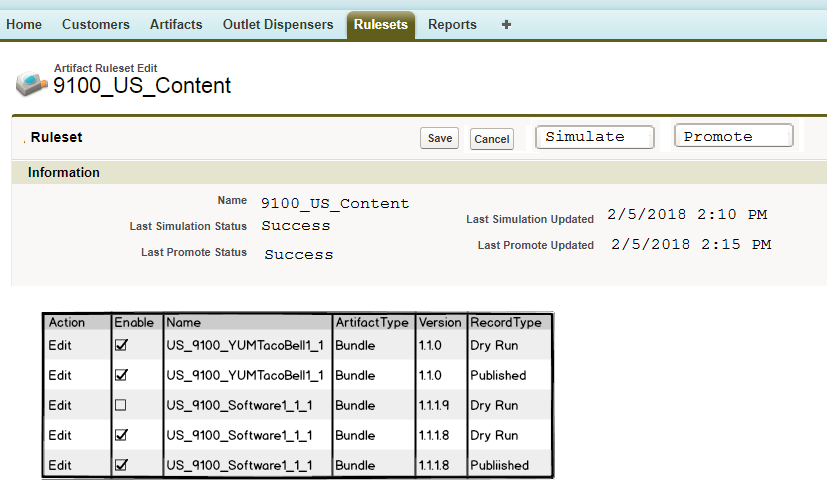


Fig 7

**BR.FOT.UI.R.2** Last Simulation Status, Last Promote Status, Last Simulation Updated, and Last Promote Updated can never be edited from the UI. These fields are update by the integration layer.

**BR.FOT.UI.R.3** Name can only be updated during creation time.

\*\* Note in the vast majority of cases Ruleset will not be created from the UI, but instead from the API calls to FIRM.

**BR.FOT.UI.R.4** Users can click on “Simulate”. When the user clicks Simulate, FOT will call API A3 to start the simulation. See FDM.A3.docx for details.

After the user clicks simulate, the user will have to refresh the screen F5 to see the updated status.

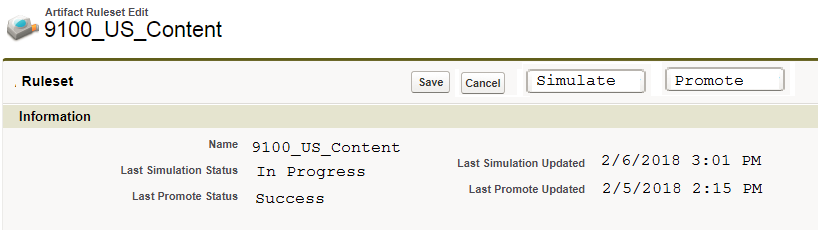


Fig 7

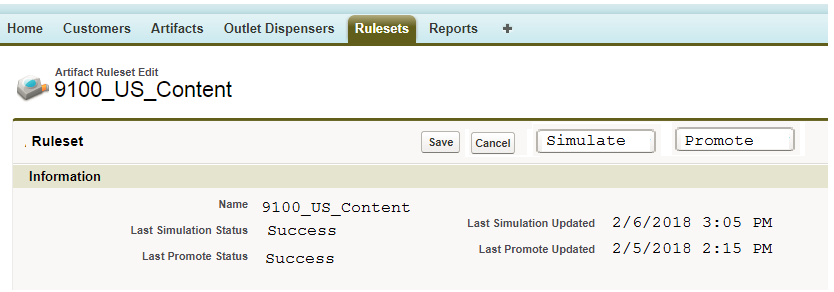


Fig 8

Once the simulation is complete the user can check the reports section to view the results of the simulation.

**BR.FOT.UI.R.4** Users can click on “Promote”. When the user clicks Promote, FOT will call API A14. A14 is the Promote PreFlightCheck API.

**BR.FOT.UI.R.4** If A14 returns a message indicates it is valid to promote like the json below, then FOT will present a message box like in Fig 9 below

\*\* Note the message is multiline and will be base64 encoded.

{

"valid": "True",

"hashCode": "57587421",

"message": "Warning this will promote 3 Artifacts Changes and impact 3561 Dispenser/OD records.

Summary of dispenser changes:

Old Count New Count

US 9100 - Wendys1.2.0 3300 US 9100 - Wendys1.2.1 3300

US 9100 - FHS1.1.7 261 US 9100 - FHS1.1.8 261 "

"textArea":{

"Artifacts to be added":

{

"id": "akjbga;gejbewjt",

"uuid": "akgdsagjdsagdsgbdsjg",

"name": "US 9100 - Wendys1.2.1",

"version": "1.2.1"

},,

"Artifacts to be updated":

{

"id": "gehgewgihewgewkndsndsds",

"uuid": "akgdsagjdsagdsgbdsjg",

"name": "US 9100 - FHS1.1.8",

"version": "1.1.8"

"deploymentGroupRuleFrom": "serailNumber = 'ZPL123456A'"

"deploymentGroupRuleTo": "outletState = 'GA'"

},

{

"id": "gehgewgihewgewkndsndsds",

"uuid": "akgdsagjdsagdsgbdsjg",

"name": "US 9100 - FHS1.1.8",

"version": "1.1.8"

"deploymentGroupRuleFrom": "serailNumber = 'ZPL123456A'"

"deploymentGroupRuleTo": "outletState = 'GA'"

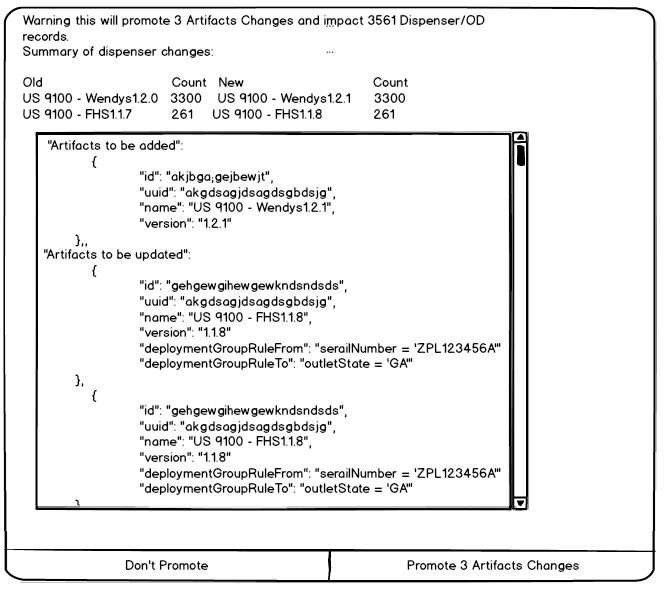
}

}

"noMsg":"Don't Promote"

"yesMsg"; "Promote 3 Artifacts Changes"

}



\*\* Notice that text for no on the left and yes on the right come from the message response sent from FIRM.

**BR.FOT.UI.R.5** If the user enter clicks on the “No” message on the right hand side then FOT should not call A3 to promote.

If the user clicks on the “Yes” message on the right hand side, FOT should call A3 to promote. See FDM.A3.docx for details.

Note FOT receives a hashCode in the A14 response, FOT must pass the hashcode the A3 command to promote. If any data change between the time of A14 and A3 then the command for A3 will fail and no artifacts in salesforce will be updated.

After the user clicks “yes” message on the right hand side, the user will have to refresh the screen F5 to see the updated status.

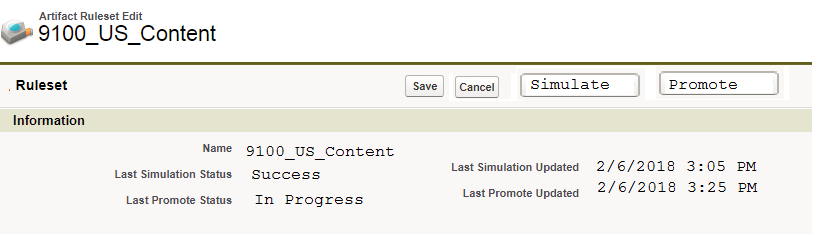


Fig 9

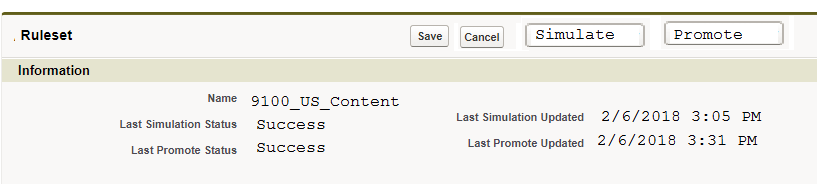


Fig 10

Once the promote is complete the user can check the reports section to view compliance status of the promote.

**BR.FOT.UI.R.4** If A14 returns a message indicates it is not valid to promote like the json below, then FOT will present an error message box with the message and the user will not be allowed to promote

{

"valid": "false",

"hashCode": "invalid",

"message": "Failed with sql exception Blah Blah"

"textArea":""

"noMsg":"QA enter bug for FOT if you see this"

"yesMsg"; "Qa enter bug for FOT if you see this"

}

\*\* Note unless someone, edits a rules salesforce database, a developer deploys bad integration code, bad settingtemplates this should not happen. This is a failsafe check to make sure we don’t publish invalid rules.

1. The Reports Tab

NOTE the reports section here is meant to describe the most critical reports and nuance of how they need to be display. It is not meant to cover all of the reporting requirements that may existing in Jira.

User will have at least three canned reports. These canned reports can each be cloned and customized.

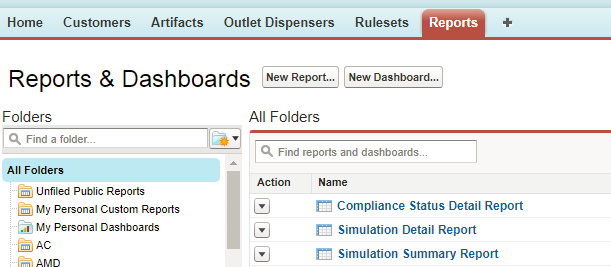


Fig 11

Once the user completes and simulation the next step for the user would be to look at the “Simulation Detail Report” and the “Simulation Summary Report” to determine if they like the results of their simulation.

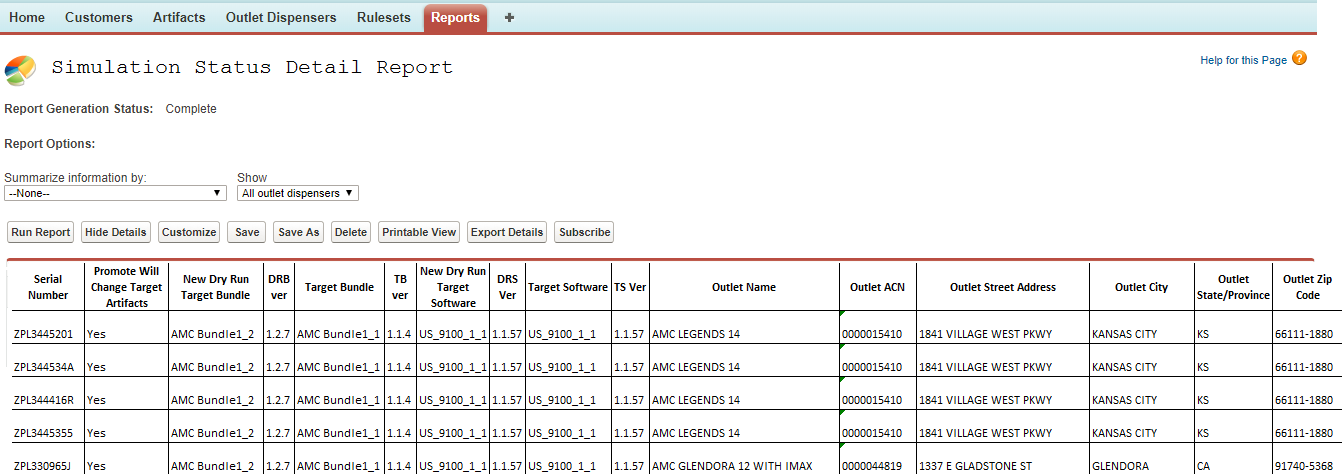


Fig 12

Full results of the tabular data can be found in SimulationDetailReportExample.xlsx.

**BR.FOT.UI.R.5** Providing the report show in Fig 12 is mandatory.

Promote Will Change Target Artifacts is mapped to PromoteWillChangeTargetArtifacts setting value.

Rule Engine Message is mapped to DryRunRuleEngineMessage setting value

**BR.FOT.UI.R.6** This report compares data with RecordType “Dry Run” and “Target” Target Artifacts in the same row.

**BR.FOT.UI.R.7** The solution must be capable of comparing “Dry Run”, “Target”, “Actual”, “Rollback” in ad hoc reports where data from the “Dry Run”, “Target”, “Actual”, “Rollback” are all in the same row.

In all Jira requirement requesting to provide the “Dry Run”, “Target”, “Actual”, “Rollback” comparisons for a dispenser. The intention is to provide the data in the same row as shown in Fig 12 and 14. **Separate rows are not acceptable.**



Note the integration layer is responsible for updating targetId, actualId, dryRunID, and rollbackId to make this possible.

TO DO Todd write up Simulation Summary Report.

Once the user completes and promote the next step for the user would be to look at the “Compliance Status Detail Report” to track the progress of the individual installs.

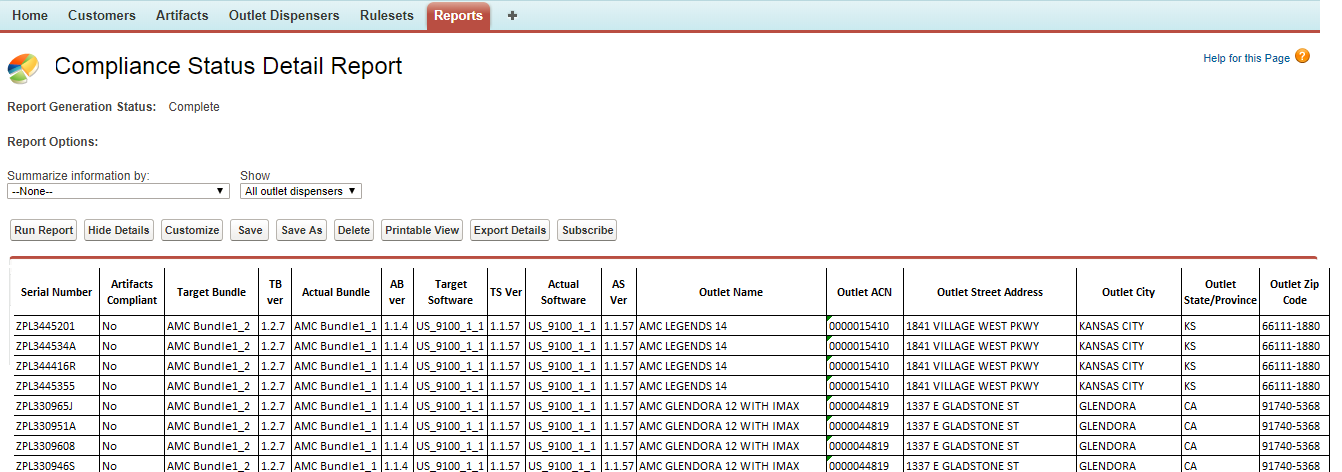


Fig 14

Full results of the tabular data can be found in ComplianceStatusDetailReportExample.xlsx.

**BR.FOT.UI.R.8** Providing the report show in Fig 12 is mandatory.

Artifacts Compliant is mapped to ArtifactsCompliant setting value.

Rule Engine Message is mapped to RunRuleEngineMessage setting value

TO DO Todd Write chart Ad Hoc example.

.

TO DO Matt and Faisal should we have?

* ArtifactsCompliant
* BundleArtifactsCompliant
* SoftwareArtifactsCompliant

TO DO Matt and Faisal should we have?

* PromoteWillChangeTargetArtifacts
* PromoteWillChangeBundleTargetArtifacts
* PromoteWillChangeSoftwareTargetArtifacts
* PromoteWillChangeSettingTargetArtifacts

1. The OD Record Tab

TO DO Todd write up the OD record tab

1. Change the to the xml for realtime integrations.

TO DO Todd write up

**End** of Summary to Salesforce UI Requirements as outcome of 2/7/2018 meeting

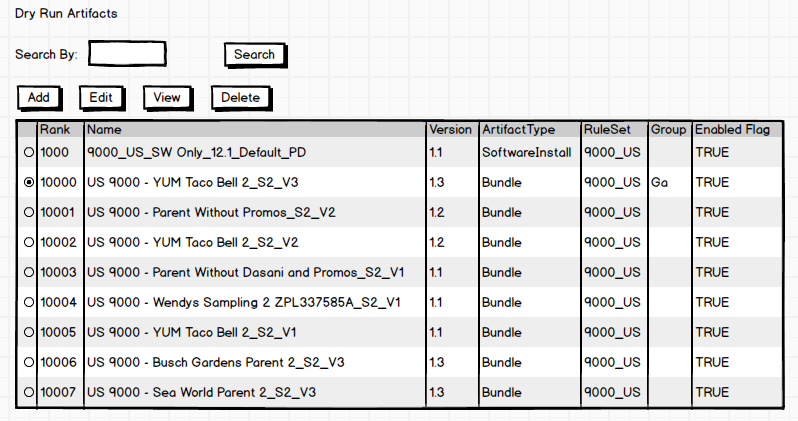
The below UI screen shots have been refactored to match above Summary.

**P1.S1:** Operations uses CMS to publish an AEM bundle. CMS Publishes the AEM bundle to S3 as a .pkg file. CMS uses API A1 and A2 to publish the artifacts to the Dry Run tables in Salesforce. See the appendix for details on APIs A1 and A2.

Example data for artifacts that can be published can be found in ArtifactsExamples.csv

**P1.S2:** Dispenser Developers use Jenkins to deploy builds. Jenkins build scripts will copy software to S3 as a .pkg file. Jenkins will call APIs A1 and A2 to publish artifacts to Dry Run tables in Salesforce.

**P1.S3.1:** Operations use the Salesforce UI to enable artifact rules.



TO DO Todd update screen shot for group

**BRP1.S3.1.2** Search applies to all of the fields below

Name

Version

ArtifactType (Miscellaneous)

RuleSet

Group

Wild cards implicitly apply for example “Wendys” means like ‘%Wendys’ or like ‘Wendys%’ or like ‘%Wendys%’ or like ‘Wendys’

Fig P1.5

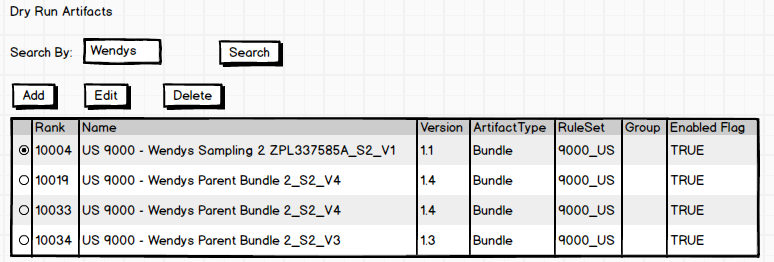


Fig P1.6

CRUD Artifacts Screen.

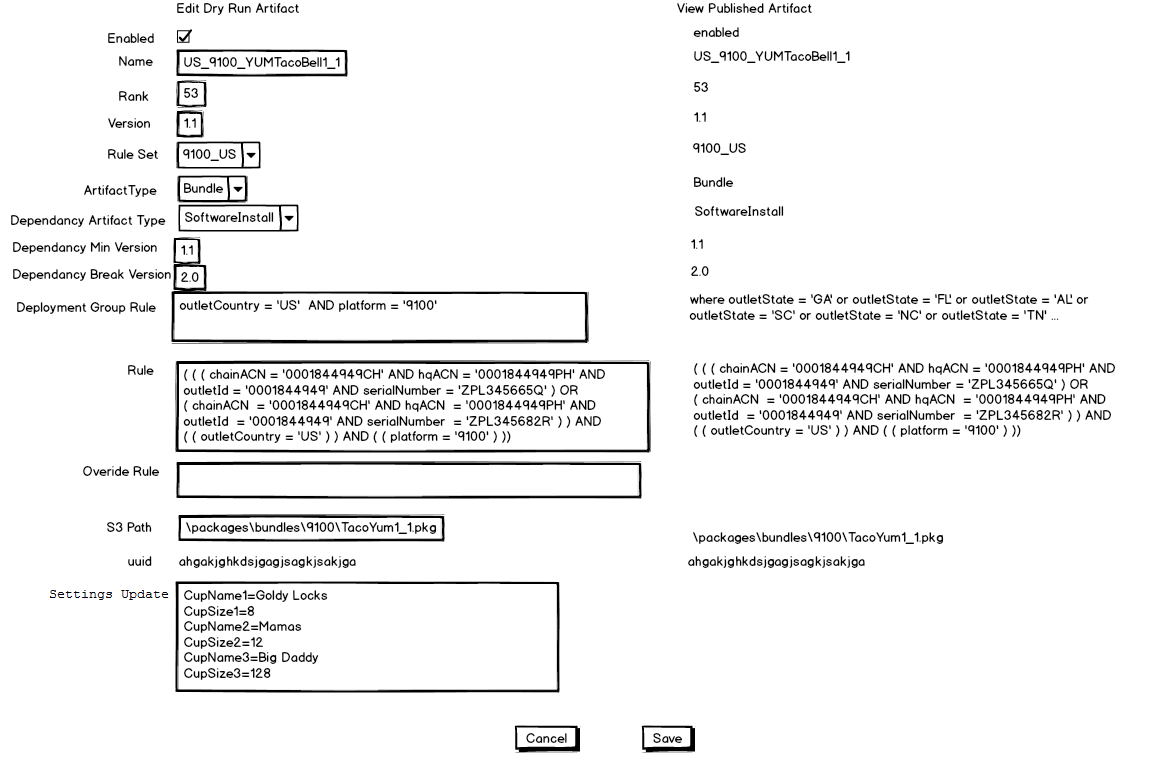


Fig P1.7

**BR.P1.S3.1.1** (Deprecated)

**BR.P1.S3.1.2** The list of ArtifactTypes and DependancyArtifactTypes to be driven by “ArtifactType.csv”

By default both ArtifactTypes and DependancyArtifactTypes should be unselected for new aritifacts.

**BR.P1.S3.1.3** When the user clicks save in the screen the rule and all other fields will be validated by calling API A5. If the rule is deemed invalid by the return of the API then the user cannot save.

See A5 Brs for validation logic.

**BR.P1.S3.1.4** If the rule and all other fields are valid the number of records matching the combination of the “Rule” or “Override Rule” (if there is an override rule the override rule is used instead of the rule) and “Deployment Group Rule” will be shown to the user on screen when the record is saved.

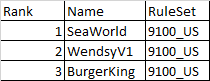
**BR.P1.S3.1.5** If an artifact (DryRunArtifact) is saved and it has a conflicting rank within the ruleset other artifacts in the ruleset then the ranks of other artifacts in the rule set will be bump to a higher number

**BR.P1.S3.1.6** Users cannot edit the UUID of an artifact in salesforce. If the user changes the S3 path for an artifact then the Salesforce will set the UUI to null.

Why? FIRM will reset the UUID based on the filed content when this happens during the next simulation or promote.

For example:

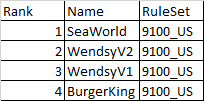
Precondition:



Then save

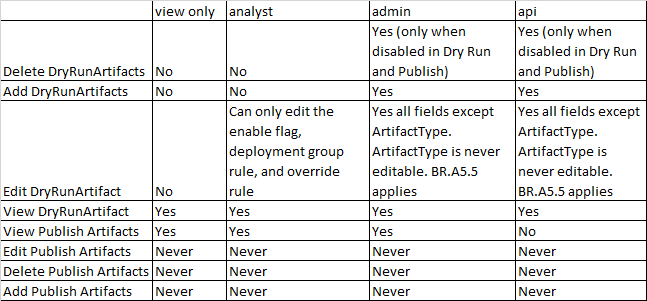
WendysV2 with Rank of 2

Post Condition:



**BR.P1.S3.1.7** There will be three user roles “view only”, “analyst”, and “admin”

The following privileges apply:



**(Note read only screens to view the Artifacts tables are required in FET)**

Once operations is satisfied with enabling the rules for artifacts operations will click a button to “Simulate Dry Run Rules”.

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>New Screen Shot>>>>>>>>>>>>>>>>>>>>>>

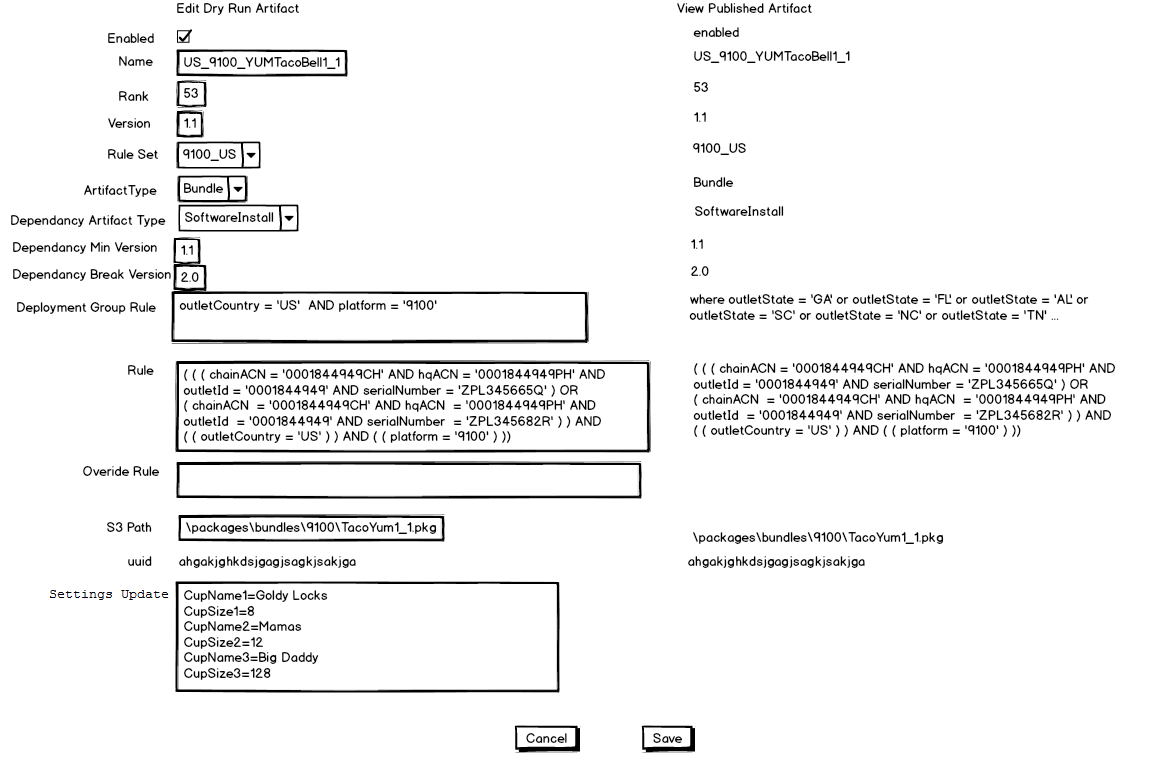


Fig P1.8

**BR.P1.S3.1.7** If the user select “Setting” artifact the s3path and version will disappear and the new text area will appear for the user to enter settings.

**P1.S3.2:** Clicking “Simulate Dry Run Rules” Sales force will take the following actions:

-P1.S3.2.2 Call API A3 to trigger the FIRM “Simulate Dry Run Rules” Job.

-P1.S3.2.1 When sales force gets a 202 response it will update DeploySimStatus table and the set the row named “Simulation” to “In Progress”



**P1.S3.3:** FIRM Simulate Job

**P1.S3.3.1**

The firm job to will effectively read the full set of all OD records and all setting from FET (this is accomplished by reading the settings table)



Fig P1.11

**P1.S3.3.2**

Once the in memory list of dispensers has been built … Next, the Firm Job will read the data in DryRunArtifacts table for the specified ruleSet in Sales Force

TO DO Todd remove write of DryRunArtifacts.json … this is no longer in the design.

The rules engine will use the dispenser data from BR S3.3.1.x BRs and the merge of data for the rule set being promoted from DryRunArtifacts to Artifacts table to calculate the DryRunTargetArtifacts.



Fig P1.12

Operations can view the results in Ad hoc reporting in Salesforce

TO DO Todd update the prototype code to handle dependency rules.

TO DO Stanley write all the rules for the rules engine based on prototype code.

**P1.S3.3.3** Finally, save the dry run artifacts to the database in FET. That is the DryRunTargetArtifacts table will be populated with the results from of the rules engine.

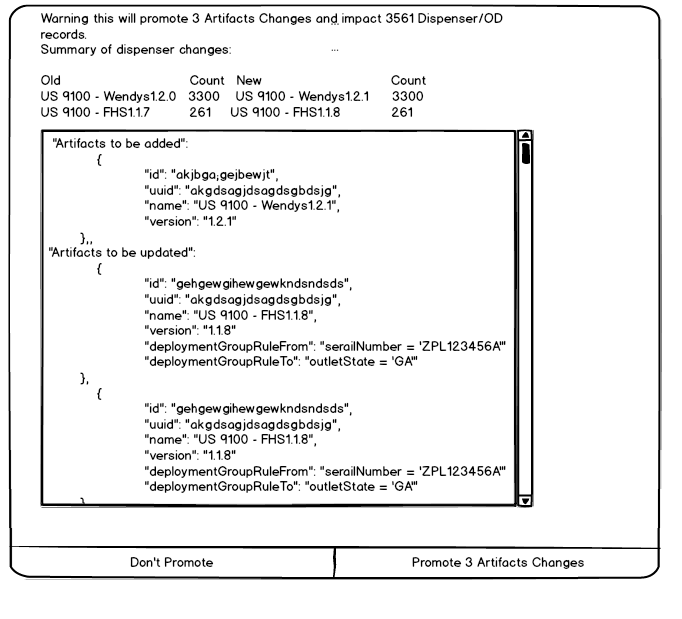
**P1.S3.3.4** Once all the DryRuTargetArtifacts have been saves the FIRM job will update DeploySimStatus table to successful with timestamp



**P1.S4.1** Operations user pushes the button in Salesforce to “Promote Artifacts” for a ruleset.

**P1.S4.2** This results in Salesforce taking the following actions:

-P1.S4.2.1 Call API 14 to perform a PromotePreFlightCheck and the user in Sales force will get a message like the below and choose if they want to proceed.



-P1.S4.2.2 Call API A3 to trigger the FIRM “Promote Dry Run Artifacts” Job.

-P1.S4.2.3 When sales force gets a 202 response it will update DeploySimStatus table and the set the row named “Promote” to “In Progress”



If we successfully perform all the actions in “Simulate Dry Run Rules” Job without error, then we proceed to execute S4.3 otherwise the job fails and we update DeploySimStatus with failure.

**P1.S4.3** Merge DryRunArtifacts to Artifacts for the rule set in Sales Force

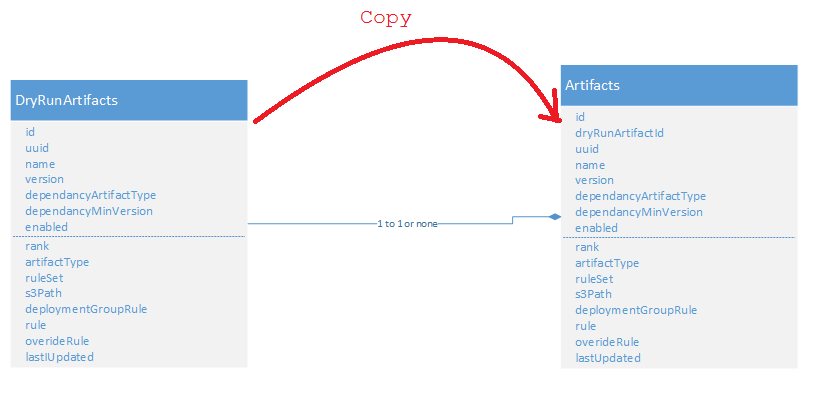


Fig P1.13

**P1.S4.4** Save the artifacts.jon to S3

**P1.S4.5** Call API A12 to notify device management of all artifacts

**P1.S4.6** Save the target artifacts to the database in FET. That is the TargetArtifacts table will populated with the results from of the rules engine.



Fig P1.14

**P1.S4.7** For all dispensers where the target state was changed call API A4 to notify device management of all server side settings and target state artifacts.

Finally, we update DeploySimStatus table.



# Calculate the content of and Job for a dispenser. A Job is an atomic update for a dispenser (an atomic update could mean content bundle install, content bundle removal, and/or software update)



Fig P2.1

**P2.S1:** FIRM Calls API A4 described in problem #1 above, providing settings and TargetStateArtifacts

**P2.S2:** Device Management passes the settings and the target state from the from A4 on to the dispenser in A6.

**P2.S3:** Device Agent sends the actual state of artifacts deployed to the dispenser.

**BR.P2.2.S3.1** If the actual state of artifacts on the dispenser differs from the target state passed to the dispenser when API A6 is called, the dispenser must call API A7 to pass settings and the actual stage back to device management.

If the target state and the actual state of artifacts differ we proceed to step S4 below.

**P2.S4:** Device management reads the manifest of all artifacts.

**P2.S5:** Device management reads all \*.pkg files for any target artifacts that must be deployed to the dispenser

(Note S6 was removed because it is not needed)

**P2.S7:** Device management download and install the package of on the dispenser.

**P2.S8:** The device agent calls API A7 again to confirm that content has been installed.

**BR.P2.S8.1** Confirmation that content is installed does not just mean file delivery. The agent code must interact with the dispenser code and confirm that the package is really installed.

TO DO Stanley Write up Sequence Diagram for how changing the target state can cancel a job.

## Enrollment/Un-Enrollment



**Fig P3.1 Enrollment Part I Steps S1 – S5**

**P3 S1 and S2:** The technicians pick the appropriate outlet or OD record.

The implementation of S1 and S2 here falls entirely on coke dispenser and integrations team.

TO DO Todd and Stanley … write the details for this

See API A8 for context.

**P3.S3:** Once the technician has selected the correct OD record from the dispenser in S2. The prepare for enrollment API A8 is called

**P3.S3.1** Call API A9 with using the json from A8 for the dispenser the technician selected.

**P3.S3.2** Coke lambda function calls the device managementimplementation of A9.

**P3.S3.3** If the call to device management succeeds coke implementation of A9 then creates an OD record in Salesforce if needed.

**P3.S3.4** Coke implementation of A9 logs all details of the transaction to FIRM database

Coke implementation of A9 returns the result to the dispenser provided by device management.

**P3.S4.1** Device Agent calls API A10 to enroll the dispenser with whatever was passed to make to dispenser in in A9.

**P3.S4.2** Device management creates a new cert that is specific to the device and returns the path for the new cert to the device as part of the response to API A10

**P3.S5** The device downloads and installs the new cert



**Fig P3.2 Enrollment Part II Steps S6 – S9 (The reader should notice this has overlap and looks similar to problem #2)**

**P3.S6** After downloading a new device specific cert. The dispenser agent calls A8 passing the serial number and outlet ACN.

Not the response for A8 looks just like a single record for A4 and the dispenser is the caller.

**P3.S7** The device agent will call API A7 to send all of its settings and actual state of artifacts deployed to dispenser to device management.

This call will include:

To Do Todd update with new json

“EnrolledStatus”: “Enrolled”

Device management will take the payload from the device and pass it on to the coke implementation of API A7.

**P3.S8** Let the technician know that he has successfully connected to device management and it if any new content or software needs to be downloaded.

**P3.S9**  The call to A7 may trigger new jobs to the dispenser.

The full enrollment sequence.



TO DO Stanley Write up un-enrollment sequence

# Synchronize settings between the dispenser and FET, report on settings.

**But first it’s useful to review the current state of synchronizing settings mastered on the server side to device management**.



**CS1:** Today when an OD record changes in FET (Salesforce) this triggers a call to API A11.

**CS2:** An aws lambda function writes this request to FIRM db

**CS3:** AWS Lambda function returns 202 processing

**CS4:** FIRM makes a lot of API call to update Air Watch … between 7 – 16 API calls to Air Watch per call to A11 … sometimes this process can take minutes ☹

**CS5:** FIRM updates the OD record in FET with 200 to indicate success.

There are a variety of integrations to get the enrolled status and setting back from dispenser to FET.

**Target architecture for pushing settings to device management from FET.**

There will be a new table in sales force with all of the settings for a dispenser that will be controlled by operations.



**Fig P4.1**

There will be a manifest of all allow settings in Freestyle. See SettingTemplates.csv for incomplete example data. It is highly recommended that the reader stop reading now and review the example data before moving on to reads the rest of the document.

Currently for the coke integrations team there are currently 2 similar files in FIRM CAMapping.csv used in Firm Real-time and FieldMapping.csv used in FET – AW sync. In the target state both of these files will be merged to form SettingTemplates.csv.



**Fig P4.2**

**P4.S1.2** FET will call API A11 to notify FIRM of the change.

**BR P4.4.S1.2** If either the OD record table or the Setting table is update this will trigger API A11

**P4.S2:** An aws lambda function writes this request to FIRM db

**P4.S3.1:** AWS Lambda function returns 202 processing  
**P4.S3.2** FIRM reads settings and OD record from sales force to create a full OD record with all settings.

**P4.S3.3** FIRM reads all artifacts that have been deployed to production … (from S3 reading from salesforce is too slow)

**P4.S4.1:** FIRM formats the JSON to make the API call A4 to device management passing

-all of the settings where MDM is OPEN or SERVER\_MASTER in settingtemplates.csv

-the target state artifacts for the device

FIRM checks the S3 cache of the last API call firm made to A4 for the given serial and outlet ACN

If FIRM determines that the json to be sent to API A4 is different from the cache then it proceeds to S4.2

Why? This prevents a dead loop of system messaging.

**P4.S4.2:** FIRM call API A4 with updated JSON

It’s worthy to take not that P1.S4 here is almost exactly like P1.S4.5 in section 1 (problem #1 calculating target state) and that calls to API A4 can trigger a job to get new artifacts to the dispenser.

**P4.S4.3:** FIRM updates the S3 cache of the last call to A4

**P4.S4.S5:** FIRM updates the OD record in FET

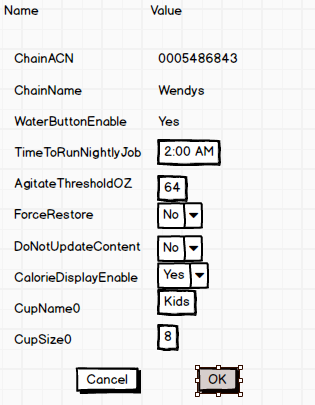
-with 200 to indicate success.

-Updates the target state if the target state changed.

**P4.S4.S6:** Device management calls APPI A6 passing on the settings and target state to the dispenser… note this is also covered in section 2 (problem #2)

**Additional settings management UI in sales force.**

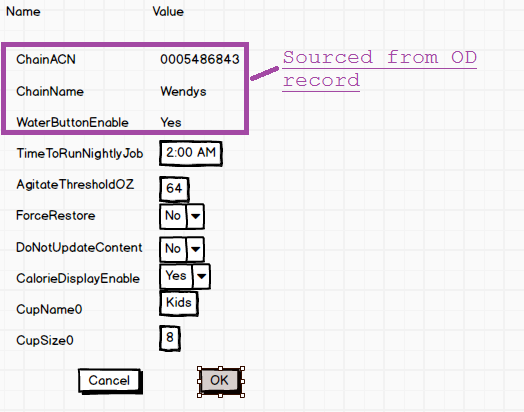
**BR.P4.UI.1** Users must be able to enter a serial number to search for a matching OD records. User must be able to select an OD record matching a serial number. Once the user selects the OD record the user must be able to navigate to a screen like the below.



**BR.P4.UI.2** All the values in both the OD Record table and the settings table that are mapped in SettingTemplates.csv will appear on the screen.

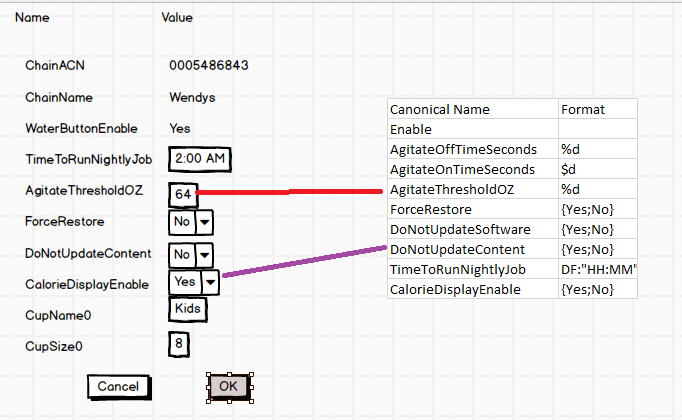
**BR.P4.UI.3**  Default values for any setting sources from the settings table will come from SettingTemplates.csv

**BR.P4.UI.5** Only settings where (the MDM field is SERVER\_MASTER or OPEN) that are sourced from settings table (controlled by settingtemplates.csv) can be edited on screen.

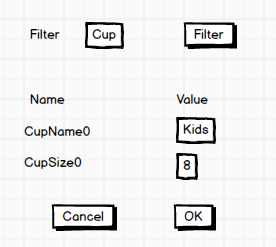


**BR.P4.UI.6** The controls and validation of fields on entry will be controller by the format field in settingTemplates.csv

Note at the time of this writing the format field is conceptual. The content in this field can be adjusted to meet the design goal and ease implementation in sales force.



**BR.P4.UI.7** There will likely be many setting for a dispenser. The user must have method to filter the settings for a dispenser.



**Target architecture for pushing settings to salesforce from the device.**



**Fig P4.3**

**P4.1.S1:** Crew member changes a cup size or swaps a pibb cartridge for pepper.

**P4.1.S2:** Device Agent calls API A7 with the settings changes and actual state of artifacts.

**P4.1.S3:** Device management passes this call through to Lambda functions coke implementation of A7

**P4.1.S4:** Coke implementation of A7 reads the SettingTemplates.csv

**P4.1.S5:** Coke implementation of A7 updates setting table and OD record in salesforce with setting that are device sourced.

**P4.1.S6:** Change to the setting and OD record in Sales force will trigger an update the FIRM calling API A11.

**P4.1.S7.1** Firm will read the settings from sales force and formats the JSON to make the API call A4 to device management passing

-all of the settings where MDM is OPEN or SERVER\_MASTER in settingtemplates.csv

-the target state artifacts for the device (calls rules engine code to calculate)

**P4.1.S7.2** FIRM checks the S3 cache of the last API call firm made to A4 for the given serial and outlet ACN

If FIRM determines that the json to be sent to API A4 is different from the cache then it proceeds to S7.3

Why? This prevents a dead loop of system messaging.

**P4.1.S7.3** FIRM calls API A45 passing settings and target state

**P4.1.S8** Device management passes through the call from A4 to the device

**P4.1.S9** Device responds with device side settings and actual state.

## Executing a Job and tracking the status of a job.

TO DO BSquare and Candid to provide

## Reporting debugging for execution of job on the dispenser

TO DO BSquare and Candid to provide

## Installation of Job on a dispenser (means content bundle install, content bundle removal, and/or software update)

TO DO BSquare and Candid collaborate with the dispenser team on this.

Dave my suggestion is for you define APIs on your side that will allow device management to provide A6 and call A7

1. Remote control of the dispenser.

TO DO BSquare and Candid to provide solution to for SSH, remote file access, and VNC like screen sharing.

1. Other weird corner cases related to the first 8 problems

TO DO Todd and Stanley FET – AW Sync will be extended to FET – AW – DM Sync … write this part up

TO DO Todd need to write up collect CAs and Brand API implementation

## API Appendix:

Each API will have separate design doc formatted as FDM.A\*.docx. Below are just a brief description, please refer to design doc for most-up-to-date information.

|  |  |
| --- | --- |
| **A1** | *Search API for Dry Run Artifacts in FOT* |
| **Implementer** | KO Integrations |
| **Caller** | CMS and Jenkins (Dispenser Build Process) |
| **URL** | [https://baseURL/<env](https://baseURL/%3cenv)>/artifacts/list |
| **Method** | **GET** |
| **URL Params** | **Required:**  ruleSet=[String], example: ruleSet=9100\_US artifactType=[String], example: ruleSet=9100\_US |
| **Headers** | **Authorization:** Basic XXXXXXXXXXXXX |
| **Success Response** | **Code:**  204 **Content:** TO DO  **Code:** 200  **Content:**  See “A1 Example.json” |
| **Responses** | **Code:** 200 Success  **Code:** 202 No content  **Code:** 400 BAD REQUEST  *If any parameters are invalid*  **Code:** 401 UNAUTHORIZED  *If Authorization header is invalid*  **Code:** 500 INTERNAL SERVER ERROR  *If server error* |
| **Sample Call** | https://firmapi.coke.com/prod/artifacts/list?ruleSet=”9100\_US”&artifactType=”Bundle” |

**BR A1.1** The caller must supply a non-empty value for the ruleset, otherwise the caller will receive error code 400.

**BR A1.2** The caller supplies an artiFactType configured in ArtifactType.csv

**BR A1.3** If there are no records matching the ruleset and artiFactType API will return 204.

**BR A1.4** The results in the json will be sorted by ranks with highest rank (lower number means higher rank) first.

**BR A1.5** API will return results from the DryRunArtifacts table in Salesforce.

Usage conventions:

CMS will always pass artifactType=”Bundle”

|  |  |
| --- | --- |
| **A2** | *API for publishing Dry Run Artifacts in FOT* |
| **Implementer** | KO Integrations |
| **Caller** | CMS and Jenkins (Dispenser Build Process) |
| **URL** | [https://baseURL/<env](https://baseURL/%3cenv)>/artifacts/publish |
| **Method** | **Post** |
| **URL Params** | **None** |
| **Headers** | **Authorization:** Basic XXXXXXXXXXXXX |
| **Payload** | See A2 Example.json |
| **Responses** | **Code:** 200 Success  **Code:** 400 BAD REQUEST  *If payload is not valid*  **Code:** 401 UNAUTHORIZED  *If Authorization header is invalid*  **Code:** 500 INTERNAL SERVER ERROR  *If server error* |
| **Sample Call** |  |

**BR.A2.1** Successful calls tothis API upserts the data provide to the DryRunArtifactsTable in Salesforce.

**BR.A2.2** Passing a null or missing id indicates that the API should add a record in Sales Force

artifacts": [{

"uuid": "TO DO insert UUID from .pkg file",

"name": "US 9100 - YUM Taco Bell 2\_S2\_V3",

"version": "1.1",

**BR.A2.3** Passing an id indicates that the API should update a record in Sales Force

"artifacts": [{

"id": "a45mn43224bkj5j4njtqkjnknt",

"uuid": "TO DO insert UUID from .pkg file",

"name": "US 9100 - YUM Taco Bell 2\_S2\_V3",

**BR.A2.4** All the validation rules in A5 below apply.

**BR.A2.5** All artifacts must have the same ruleset

**BR.A2.8** If any artifact in the rule set in not included in the post. That artifact will be deleted from DryRunArtifactsTable in Salesforce.

For example, if there are 4 entries in the DryRunArtifactsTable A, B, C, D then we get a Call with A, B, C then D will be deleted.

**BR.A2.9** Whenever artifacts are updated the enabled flag is not changed.

**BR.A2.10** If the called tries to remove an entry from the DryRunArtifactsTable and either the DryRunArtifactsTable or the ArtifactsTable entry is enabled. The delete will fail.

**BR.A2.11** Call will return any records inserted into DryRunArtifactsTable as json payload of return response.

**Usage conventions:**

CMS will call API A1 like:

[https://firmapi.coke.com/prod/artifacts/list?ruleSet=”9100\_US”&artifactType=”Bundle](https://firmapi.coke.com/prod/artifacts/list?ruleSet=)”

and get a list of artifacts then add or update artifacts and re-rank them and post them back to:

API A2

CMS should post with a rank starting at 100

Ranks only have meaning within a ruleSet

|  |  |
| --- | --- |
| **A3** | *command to simulate or promote the artifacts or rules as described in Problem 2 above.* |
| **Implementer** | KO Integrations |
| **Caller** | Salesforce |
| **URL** | [https://baseURL/<env](https://baseURL/%3cenv)>/artifacts/comand |
| **Method** | **Get** |
| **URL Params** | cod allowed values are simulate and promote  ruleset must be provided if the command is simulate  hashCode must be provided if the command is promote |
| **Headers** | **Authorization:** Basic XXXXXXXXXXXXX |
|  |  |
| **Response** | **Code:** 202 Processing is success  **Code:** 400 BAD REQUEST  *If any parameters are invalid*  **Code:** 401 UNAUTHORIZED  *If Authorization header is invalid*  **Code:** 500 INTERNAL SERVER ERROR  *If server error* |
| **Sample Call** | <https://firmapi.coke.com/prod/artifacts/comand?cmd=simulate&ruleSet=9100_US>  https://firmapi.coke.com/prod/artifacts/comand?cmd=promote&ruleSet=9100\_US&hashCode=t89yh4uhtrrhjgjg |

**BR.A3.1** cmd must be simulate or promote otherwise caller will get a 400 response

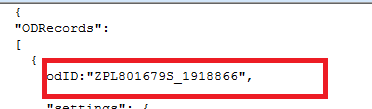
**BR.A3.2** If the user selects a simulate command the caller must supply a ruleSet. The ruleSet must match some records in Salesforce otherwise the caller will get an error.

|  |  |
| --- | --- |
| **A4** | *Accepts setting and target state from the server side cloud* |
| **Implementer** | Device Management |
| **Callers** | FIRM (Coke Integrations Team) |
| **URL** | https://baseURL/<env>/settings/pushServerSettingsToDM |
| **Method** | **Post** |
| **Payload** | See A4 Json Example.json the OD records in the json conform to ODRecord.yaml |
| **Headers** | **Authorization:** Basic XXXXXXXXXXXXX |
| **Response** | **Code:** 202 Processing is success  **Code:** 400 BAD REQUEST  *If any parameters are invalid*  **Code:** 401 UNAUTHORIZED  *If Authorization header is invalid*  **Code:** 500 INTERNAL SERVER ERROR  *If server error* |
|  |  |

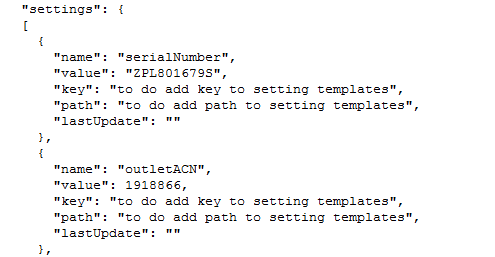
Note: ODRecord.yaml defines the OD record that is used in the Saga pattern across the ecosystem.

Implementer

**BR.Impl.A4.1** Each OD record will have an odID



**BR.Impl.A4.2** Each ODRecord in the JSON posted must contain a serialNumber and outletAcn. If this this not true device management should return 400.

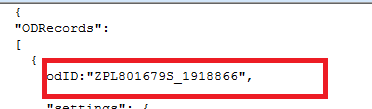


**BR.Impl.A4.3** If there is only one OD record in the request and the serialNumber and outletACN don’t match any record in device management then the API should return 204 no content.

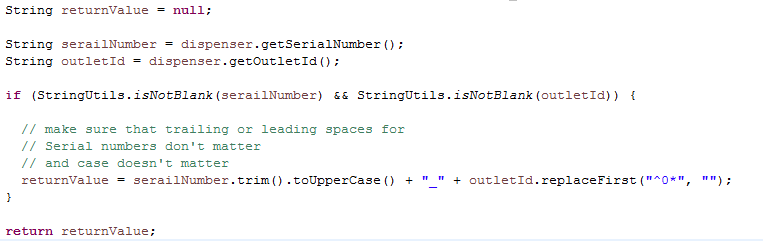
**BR.Impl.A4.4** In rare cases it is possible there could be no target state. If the DM receives a call with no target state then DM should cancel all jobs for the dispenser.

Caller:

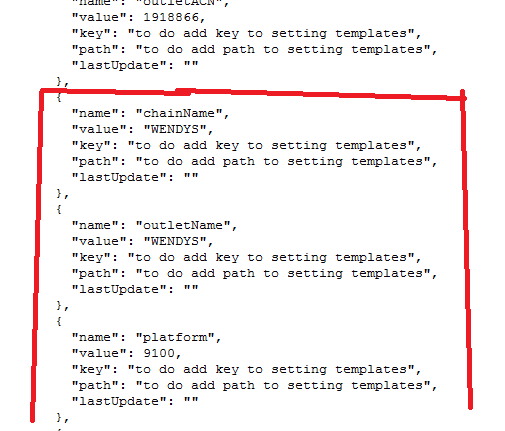
**BR.Caller.A4.1** Firm will set the odID to the upper case serial number + “\_” + the outlet ACN stripped of leading zeros.



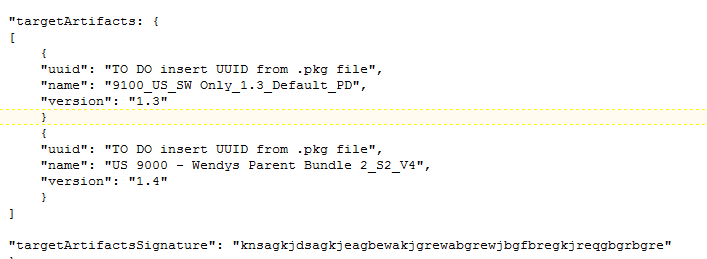
>>>>



**BR.Caller.A4.2** FIRM will include all of the settings where MDM is set to SERVER\_MASTER or OPEN (in SettingTemplates.csv) for each OD record



**BR.Caller.A4.3** FIRM will include all of the Target State Artifacts and the digital signature as shown below



|  |  |
| --- | --- |
| **A5** | *Validate artifacts before saving in Sales Force* |
| **Implementer** | KO Integrations |
| **Caller** | Sales Force |
| **URL** | https://baseURL/<env>/artifacts/validate |
| **Method** | **Post** |
| **Headers** | **Authorization:** Basic XXXXXXXXXXXXX |
| **Payload** | See A5 Payload Example1.json |
| **Response** | **Code:** 200 is success  See: A5 Success Response. Json  See: A5 Failed Response.json  **Code:** 400 BAD REQUEST  *If the json is not properly formed*  **Code:** 401 UNAUTHORIZED  *If Authorization header is invalid*  **Code:** 500 INTERNAL SERVER ERROR  *If server error* |
|  |  |
|  |  |

**BR.A5.1** The following validation rules apply for artifact validation.

**enabled:** must not be empty  
**name:** If the id passed is null the name should not match any existing DryRunArtifact in Sale Force matching the same ruleset and artifact Type(to prevent create of duplicate names)

**rank:** must not be empty and must be a positive integer.

**version:** must be a string with this, this.that, or this.that.theother notation (this that and the other are all number)

**artifactType:** must be configured in ArtifactType.csv in S3

**ruleset:** cannot be null or empty.

**dependancyArtifactType:** Can be empty or a type configured in ArtifactType.csv in S3

**dependancyMinVersion:** if dependancyArtifactType is not null must be a string with this.that or this.that.theother notation (this that and the other are all number)

**dependancyBreakVersion:** if dependancyArtifactType is not null must be a string with this.that or this.that.theother notation (this that and the other are all number)

**s3Path:** Path must be reached and must contain valid .pkg file.

If the uuid in sales force table is not null, the uuid of the package file must match the uuid in Salesforce Dry Run Table.

**deploymentGroupRule:** If the enables is set to the true this must not be empty or null.

**deploymentGroupRule, rule, and overrideRule:** These must be combined and tested by the rules engine for validity.

Rules must use the “Canonical Name” from SettingTemplates.csv … rules will be validated any rule that does not compile will cause publishing to fail.

**BR.A5.2** No artifact in the same rule set may have the same name

**BR.A2.3** If the UUID of the artifact changes the version must change for the call to succeed.

**BR.A5.4** API will return the number of records select form the last cache of dispenser info from FET.

**BR.A5.5** If the called tries to inappropriately update an entry in the DryRunArtifactsTable and either the DryRunArtifactsTable or the ArtifactsTable entry is enabled. The update will fail.

There will be a config setting

artifactFieldsNotAllowedToUpdateWhenEnabled=name,uuid,version,s3Path,settingsContent

if any fields in artifactFieldsNotAllowedToUpdateWhenEnabled change the artifact is enables it is considered an inappropriately update

TO DO Todd Add check artifactType can never update.

|  |  |
| --- | --- |
| **A6** | *Dispenser accepts setting and target state from the server side cloud* |
| **Implementer** | Device (Device Management Vendor Provides Agent Implementation) |
| **Caller** | Device Management |
| **URL** | Will likely be an MQTT message … device management vender to decide |
| **Method** | Will likely be an MQTT message … device management vender to decide |
| **Payload** | The payload is conceptually a pass through of a single ODRecord from A4 … See A6 Json Example.json |
|  |  |

TO DO BSquare and Candid provide draft java interfaces to-from FOS and java agent.

**BR.Impl.A6.1** Dispenser agent will notify FOS application of any settings that have been changed where MDM is SERVER\_MASTER. FOS application will save the settings and make sure that they take effect.

**BR.Impl.A4.1** Dispenser agent will provide the target artifacts to FOS and FOS with provide a list of missing artifacts in A7

Device Management and Coke Integrations Team both product and implementation of this API

So you will A7 listed twice

|  |  |
| --- | --- |
| **A7** | *Accepts setting and actual artifact state from device* |
| **Implementer** | Device Management |
| **Callers** | Device Agent |
| **URL** | Will likely be an MQTT message … device management vender to decide |
| **Method** | Will likely be an MQTT message … device management vender to decide |
|  |  |
| **Payload** | See A7 Json Example.json |
| **Sample Call** |  |
|  |  |
|  |  |

**BR.Impl.A7.1** The json payload will contain all of the setting that can be edited on device.

**BR.Impl.A7.2** The json payload will contain the actualArtifactsRunning. For example, the dispenser could be running a Wendy’s bundle, but a mope’s bundle has been downloaded but not yet installed. So, in this case actualArtifactsRunning includes Wendy’s.

**BR.Impl.A7.3** The json payload will contain removed Artifacts. The removed artifacts should only be included if this is the call following an install where an artifacts was previously running and is no longer running.

**BR.Impl.A7.4** The The json payload will contain missingArtifacts if FOS says its missing artifacts.

|  |  |
| --- | --- |
| **A7** | *Accepts setting and actual artifact state from device* |
| **Implementer** | Device Management … and Coke integrations both produce an implementation of this API. |
| **Callers** | Device Agent (Written by device management vender) calls device management  Device Management calls Coke integrations implementation. |
| **URL** | Not necessarily http … must be highly secure with a unique cert for each dispenser. |
| **Method** | **Actual Method left to device management implementer** |
|  |  |
| **Payload** | TO DO Todd payload has serial outletACN all settings and ActualArtifactState |
| **Sample Call** |  |
|  |  |
|  |  |

TO DO Todd Write the prototype code for this API to update salesforce

|  |  |
| --- | --- |
| **A8** | *Search for enrollment records* |
| **Implementer** | Coke integrations team |
| **Caller** | Dispenser written by coke dispenser team. |
| **URL** | https://baseURL/<env>/enrollment/search |
| **URL Params** | TO DO Stanley callers can search by serial and platform, outlet id, or address |
| **Response** | **Code:** 200  See “A8\_A9\_A10 Json Example.json” for content.  **Code:** 400 BAD REQUEST  *If any parameters are invalid*  **Code:** 401 UNAUTHORIZED  *If Authorization header is invalid*  **Code:** 500 INTERNAL SERVER ERROR  *If server error* |
| **Sample Call** |  |
|  |  |
|  |  |
|  |  |

TO DO Stanley the allowed status need to be configured.

TO DO Todd write prototype code for this API

|  |  |
| --- | --- |
| **A9** | *Prepare for enrollment* |
| **Implementer** | Coke integrations team … and device management team must both implement. |
| **Caller** | Device code written by coke dispenser team calls coke integrations team implementation.  Coke integrations team implementation calls device management implementation. |
| **URL** | https://baseURL/<env>/enrollment/prepare |
| **Method** | **Post** |
|  |  |
| **Payload** | See “A8\_A9\_A10 Json Example.json” for content. |
| **Returns** | A payload from device management. |
| **Rationale** | Provide Device Management with a method to white list a dispenser |
|  |  |

TO DO Todd Write prototype code coke implementation.

|  |  |
| --- | --- |
| **A10** | *Enroll* |
| **Implementer** | Device (Device Management Vendor Provides Agent Implementation) |
| **Caller** | Device Agent (written by device management vendor) calls device management. |
| **URL** | Up to implementer |
| **Method** | **Post** |
|  |  |
| **Payload** | The payload returns from API A9 … see “A8\_A9\_A10 Json Example.json” |
| **Returns** | A path and details to download a new device specific cert. |
| **Rationale** | Give device management a chance to enroll. |
|  |  |

|  |  |
| --- | --- |
| **A11** | *FIRM Realtime xml master data from FET (salesforce)* |
| **Implementer** | Coke integrations team |
| **Caller** | Salesforce … FET |
| **URL** | TO DO Provide URL |
| **Method** | **Post, Put, Delete** |
|  |  |
| **Payload** | Same xml request that FIRM gets today from FET |
| **Returns** |  |
| **Rationale** | The existing API that FET used to push change to master data to FIRM. |
|  |  |

TO DO Todd Write up the prototype code

|  |  |
| --- | --- |
| **A12** | *Publish Artifacts to Device Management* |
| **Implementer** | Device Management |
| **Caller** | Coke integrations team |
| **URL** | Device Management |
| **Method** | **Post** |
|  |  |
| **Payload** | See A12 Example.json |
| **Returns** | **Code:** 200 is success  See: A5 Success Response.json  See: A5 Failed Response.json  **Code:** 400 BAD REQUEST  *If the json is not properly formed*  **Code:** 401 UNAUTHORIZED  *If Authorization header is invalid*  **Code:** 500 INTERNAL SERVER ERROR  *If server error* |
| **Rationale** | Tell device management about any Artifacts so the it know a mapping of artifacts uuids and an S3 path. |
|  |  |

**BR.A12.1** This API contains the full set of allowed artifacts. Device management is responsible for using this information to cleanup/remove old artifacts when they can be safely removed.

|  |  |
| --- | --- |
| **A13** | *Query OD Records* |
| **Implementer** | Device Management |
| **Caller** | Coke integrations team, Operations Team |
| **URL** | Device Management |
| **Params** | **TO DO Stanley work with Faisal to define parameters**  **We should be able to pull all records for Sync or do targeted queries** |
| **Method** | **GET** |
|  |  |
| **Returns** | **Code:** 200 is success  An array of OD records ODRecord.yaml format.  Or S3 path to and json doc with array of OD records ODRecord.yaml format.  The intent is to be able to download all of the OD records for Sync Purposes.  **Code:** 400 BAD REQUEST  *If the json is not properly formed*  **Code:** 401 UNAUTHORIZED  *If Authorization header is invalid*  **Code:** 500 INTERNAL SERVER ERROR  *If server error* |
| **Rationale** |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **A14** | *Promote Pre-Flight Check* |
| **Implementer** | KO Integrations |
| **Caller** | Salesforce |
| **URL** | [https://baseURL/<env](https://baseURL/%3cenv)>/promotePreFlightCheck |
| **Method** | **GET** |
| **URL Params** | **Required:**  ruleSet=[String], example: ruleSet=9100\_US |
| **Headers** | **Authorization:** Basic XXXXXXXXXXXXX |
| **Success Response** | **Code:**  200 **Content:** TO DO |
| **Responses** | **Code:** 400 BAD REQUEST  *invalid ruleset*  **Code:** 401 UNAUTHORIZED  *If Authorization header is invalid*  **Code:** 500 INTERNAL SERVER ERROR  *If server error* |
| **Sample Call** | https://firmapi.coke.com/prod/artifacts/promotePreFlightCheck?ruleSet=”9100\_US” |

**BR A14.1** A successful response will be returned if the promote of the ruleset will results in all valid parsable rules.

**BR A14.2** If successful a response similar to “A14 Success Response.json” will be returned.

**BR A14.2** If not successful and response with valid set false will be returns and Sale force will not allow the deployment to proceed.

See “A14 Failed Response.json”

Not message in the response needs to be multiline which is not allowed in Json.

Message must be base 64 encoded.