

Oracle® Cloud

Managing Oracle Autonomous Mobile Cloud Enterprise



Release 18.2.5

E95338-03

June 2018

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Oracle Cloud Managing Oracle Autonomous Mobile Cloud Enterprise, Release 18.2.5

E95338-03

Copyright © 2018, 2018, Oracle and/or its affiliates. All rights reserved.

Primary Author: Jennifer Shipman

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience	vi
Documentation Accessibility	vi
Conventions	vi

1 An Administrator's Roadmap to Oracle Autonomous Mobile Cloud Enterprise

Part I Configuring Oracle Autonomous Mobile Cloud Enterprise

2 Policies

Defining Policies	2-1
AMCe Policy Names	2-2
Policy Scope	2-3
Removing Policies	2-4

3 Credentials (CSF Keys and Certificates)

Managing Keys and Certificates	3-1
Configuring a CSF Key	3-2
Configuring a Web Service or Token Certificate	3-2
Configuring an SSL Certificate	3-2
Disabling SSL Hostname Verification	3-3
Adding a Token Issuer	3-3
Configuring Rules	3-4
Rule Types	3-4

Part II Managing AMCe Artifact Lifecycles

4 Understanding Lifecycles

Draft State	4-1
Published State	4-2
Making Changes After a Backend is Published (Rerouting)	4-3
Versioning	4-5
Deleting an Artifact	4-6
Moving an Artifact to the Trash	4-6
Restoring an Artifact	4-9
Restoring an Artifact from Administration	4-10
Purging an Artifact	4-11
Purging Artifacts from Administration	4-11
Artifact Lifecycles	4-12

5 Client and App Profile Lifecycle

Publishing a Client	5-1
Updating the Version Number of a Client	5-1
Creating a New Version of a Client	5-2
Moving a Client to the Trash	5-2
Restoring a Client	5-3
Managing Your Clients and App Profiles	5-3

6 Backend Lifecycle

Backend Lifecycle States	6-1
Publishing a Backend	6-1
Updating the Version Number of a Backend	6-2
Creating a New Version of a Backend	6-3
Moving a Backend to the Trash	6-3
Deactivating a Backend	6-4
Restoring a Backend	6-4
Managing a Backend	6-4

7 API Lifecycle

Custom APIs and API Implementations	7-1
Publishing a Custom API	7-2
Updating the Version Number of an API	7-3
Creating a New Version of an API	7-3
Moving a Custom API to the Trash	7-4
Restoring a Custom API	7-4

Managing an API	7-4
-----------------	-----

8 API Implementation Lifecycle

Publishing an API Implementation	8-1
Creating a New Version or Updating the Version of an API Implementation	8-2
Moving an API Implementation to the Trash	8-3
Restoring an API Implementation	8-3

9 Connector Lifecycle

Publishing a Connector	9-1
Updating the Version Number of a Connector	9-1
Creating a New Version of a Connector	9-2
Moving a Connector to the Trash	9-2
Restoring a Connector	9-3
Managing a Connector	9-3

10 Collection Lifecycle

Publishing a Collection	10-1
Updating the Version Number of a Collection	10-2
Creating a New Version of a Collection	10-2
Moving a Collection to the Trash	10-2
Restoring a Collection	10-3
Managing a Collection	10-3

Part III Reference

A Oracle Autonomous Mobile Cloud Enterprise Policies

AMCe Policies and Values	A-1
--------------------------	-----

Preface

Welcome to *Managing Oracle Mobile Cloud, Enterprise*.

Audience

This guide is intended for administrators who maintain and monitor services in Oracle Autonomous Mobile Cloud Enterprise.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Conventions

The following text conventions are used in this guide:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

An Administrator's Roadmap to Oracle Autonomous Mobile Cloud Enterprise

Welcome to Oracle Autonomous Mobile Cloud Enterprise! AMCe is a cloud-based service that provides a unified hub for developing, publishing, maintaining, monitoring, and analyzing your mobile apps and the resources that they rely on.

As **AMCe administrator**, you define policies, configure credentials, and manage lifecycles of the artifacts created by your team's developers. Use this guide to help you understand and manage these features.

- The way AMCe manages artifacts is governed by policies. For details on policies and how to define them, see [Policies](#).
- CSF keys and security certificates are unique to each instance of AMCe. For details on configuring credentials, see [Credentials \(CSF Keys and Certificates\)](#).
- Artifact lifecycles are simple but interdependent. For more information on lifecycles and tips for managing each type of artifact, see [Understanding Lifecycles](#).
- Diagnostics also provide important information for monitoring the health of your instance and its artifacts. For details about diagnostics, see Diagnostics in *Developing Applications with Oracle Autonomous Mobile Cloud Enterprise*.

Part I

Configuring Oracle Autonomous Mobile Cloud Enterprise

This part contains the following chapters:

- [Policies](#)
- [Credentials \(CSF Keys and Certificates\)](#)

2

Policies

Policies define properties for artifacts on a global or artifact-specific level. You can use policies to adjust how artifacts behave even after they are published.

Even though an artifact can't be changed after it's published, its behavior can still be affected. You have the flexibility to adjust policy values to better fit the required behavior of artifacts. For example, you might be testing that a connector's endpoints are valid. The timeout values don't matter much during the experimental phase, but when you publish the connector, realistic timeout settings are required. You can adjust the value of the `Network_HttpRequestTimeout` policy accordingly.


Only AMCe administrators can modify policies. You can modify policies when you deploy an artifact (client, backend, custom API, API implementation, connector, or collection) or by editing the `policies.properties` file directly from the Administration view. For information on how to change policy settings, see [Defining Policies](#).

In most cases, you should leave the policy's default settings. Make sure you understand the scope of a policy before you change it. When you change a policy with global scope, the value is applied to all relevant artifacts. For details on policy scope, see [Policy Scope](#).

For a list of policies including descriptions, scopes, and default values, see [Oracle Autonomous Mobile Cloud Enterprise Policies](#).

Defining Policies

Define and modify policies by exporting the current `policies.properties` file, editing the property values, and importing the modified file.

1. Click  and select **Settings** from the side menu.
2. Click **Policies** and then click **Export**.
3. Make a backup copy of the `policies.properties` file before you update it. If you find you've made unintended changes, you can restore policies to their previous state by importing the backup copy.
4. Open the exported `policies.properties` file in a text editor, edit the policies as needed, and save the file.
5. Go back to the Policies page, click **Import**, and select the modified `policies.properties` file.

Here's an example `policies.properties` file:

```

# -----
# ! DO NOT EDIT THE HEADER !
# Snapshot from: instance1
# System version: 17.4.3
# Data format: 2
# -----
*.Admin_DcsConfig={}
*.Admin_FeatureConfiguration={ "DCS"\: true }
*.Asset_AllowPurge=ALL
*.Asset_AllowTrash=ALL
*.Asset_AllowUntrash=ALL
*.Asset_DefaultInitialVersion=1.0
*.CCC_DefaultNodeConfiguration=6.10
*.CCC_LogBody=false
*.CCC_LogBodyMaxLength=512
*.CCC_SendStackTraceWithError=false
*.Database_CreateTablesPolicy=explicitOnly
*.Database_MaxRows=1000
*.Database_QueryTimeout=20
*.Diagnostics_ExcludedHttpHeadersInLogs=
Authorization, Cookie, oracle-mobile-uitooling-password, Oracle-Mobile-Social-Accee
*.Diagnostics_RequestPercentageErrorThreshold=10
*.Diagnostics_RequestPercentageWarningThreshold=1
*.Logging_Level=800
*.Network_HttpConnectTimeout=20000
*.Network_HttpReadTimeout=20000
*.Network_HttpRequestTimeout=40000
*.Notifications_DeviceCountWarningThreshold=70.0
*.Routing_DefaultImplementation=system/MockService(1.0)
*.Security_AllowOrigin=disallow
*.Security_ExposeHeaders=
*.Security_IdentityProviders=
[{"identityProviderName"\:"facebook", "properties"\:{"graphApiUrl"\:"https:\/\/gr
*.Security_IgnoreHostnameVerification=false
*.Security_TokenAssertionTimeoutSecs=390
*.Security_TokenExchangeTimeoutSecs=21600
*.Sync_CollectionTimeToLive=86400
*.User_AllowDynamicUserSchema=false
*.connector/CustomMapLocation(1.0).Connector_Endpoint=https:\/\/maps.googleapi
*.connector/CustomMapLocation(1.0).Security_OwsmPolicy=[]
*.connector/testsoapconn(1.0).Connector_Endpoint=http:\/\/www.thomas-bayer.com/a
*.connector/testsoapconn(1.0).Security_OwsmPolicy=[]
"

```

AMCe Policy Names

Policy names use the format `backendName.apiName.policyName`
(`mbeArtifactIdentity invocableArtifactIdentity.policyPropertyName`):

- `backendName` binds the policy value to a specific backend.
- `apiName` binds the policy to an API, API implementation, or connector API.

 **Note:**

When you define a policy that affects an API, you must use a fully-qualified name, consisting of the API category and the API name (with or without the version). The API category can be: `custom`, `connector`, `platform`, or `system`. For example: `myBackend.custom/myAPI(1.0).propertyName`. Policy names for API implementations don't include category types. For example, `*.myAPIImpl(1.0)`.

- `policyName` is the name of the policy (initial-capped and preceded by a category). For a full list of policies, see [Oracle Autonomous Mobile Cloud Enterprise Policies](#).

To set the scope of the policy you are defining, set the `backendName`, `apiName` and `policyName` in the following ways:

- A wildcard denoted by an asterisk (*), which binds the policy to all artifacts of the particular type.
- The artifact name alone, which binds the policy to all artifacts of the particular type with the specified name. For example, `myBackend`.
- The artifact name and version, which binds the policy to the artifact of the particular type with the specified name and version, for example, `myBackend(1.0.0)`.

Policy Scope

Policy scope defines which artifacts use the policy's settings. Policies can be set globally, by artifact type, or for a specific artifact. Some policies don't support all three scopes. For example, the `Connector_Endpoint` policy stores the endpoint URL of a specific connector, so it can only be set at the artifact level. The `Network_HttpRequestTimeout` policy affects APIs and connectors, so it can be set globally.

The combination of values in the name defines the scope for the policy.

- **Global scope:** Set both `backendName` and the `apiName` as wildcards so that the policy is applied globally.
For example: `*.*.Logging_Level=800`
- **Backend scope:** Set `backendName` to a specific backend and set the `apiName` as a wildcard so that the policy is applied to all APIs and connectors called in the context of the given backend. You make this more specific by including the version of the backend.
For example: `MyBackend(1.3)*.Logging_Level=800`
- **API scope:** Set `backendName` to a wildcard and set the `apiName` to a specific API so that the policy is applied to that particular API when called in the context of any backend. You make this more specific by including the version of the API.
For example: `*.custom/MyApi(2.0).Logging_Level=800`
- **API Implementation scope:** Set `backendName` to a wildcard and set the `apiName` to a specific API implementation so that the policy is applied to that particular implementation when called in the context of any backend. You make this more specific by including the version of the API implementation.
For example: `*.MyApiImpl(1.0).Logging_Level=800`

- **Connector scope:** Set `backendName` to a wildcard and set the `apiName` to a specific connector so that the policy is applied to that particular connector when called in the context of any backend. You make this more specific by including the version of the connector.

For example: `*.connector/MyConnector(2.2).Logging_Level=800`

- **Fully-qualified API scope:** Set the `backendName` to a specific backend and set the `apiName` to a specific API so that the policy is scoped at the fully-qualified API level whenever the API is called within the scope of the given backend. You make this more specific by including the version of the backend and the API.

For example: `MyBackend(1.3).custom/MyApi(2.0).Logging_Level=800`

- **Fully-qualified API Implementation scope:** Set the `backendName` to a specific backend and set the `apiName` to a specific API Implementation so that the policy is scoped at the fully-qualified API implementation level whenever the implementation is called within the scope of the given backend. You make this more specific by including the version of the backend and the API implementation.

For example: `MyBackend(1.3).MyApiImpl(1.0).Logging_Level=800`


- **Fully-qualified Connector scope:** Set the `backendName` to a specific backend and set the `apiName` to a specific connector so that the policy is scoped at the fully-qualified connector level whenever the connector is called within the scope of the given backend. You make this more specific by including the version of the backend and the connector.

For example: `MyBackend(1.3).connector/MyConnector(2.2).Logging_Level=800`

Removing Policies

You can reduce clutter in the `policies.properties` file by removing policies defined for obsolete artifacts.

To remove policies:

1. Click  and select **Settings** from the side menu.
2. Click **Policies** and then click **Export**.
3. Make a copy of the exported `policies.properties` file as a backup.
4. Open the file in a text editor and delete the policies you want to remove. Save the file.
5. Back on the Settings page, select **Delete all policies before import**.
6. Import the modified `policies.properties` file.

All the previous policies are deleted, leaving only the policies you imported from the modified file. If you find you've accidentally deleted the wrong policies, you can restore policies to their previous state by importing the backup copy of the file.

3

Credentials (CSF Keys and Certificates)

AMCe uses the Credential Store Framework (CSF) to manage credentials, allowing you to store, retrieve, update, and delete credentials for web services and other apps.

CSF keys are credentials that use basic authentication (user name and password) to certify the authority of users and system components.

Certificates are electronic documents used to authenticate an individual or organization. There are a few different kinds of certificates:


- **Web Service Certificates** can be trusted certificates containing only a public key or certificates that contain both public and private key information. Web Service Certificates are stored in the Oracle WSM Keystore.
- **Token (Signing) Certificates** are standard X509 certificates used to securely sign all tokens issued by a federation server. For third-party tokens, you configure Token Issuer Certificate and Signing Authority Certificate information, along with any intermediate certificates, to establish a chain of trust. For details, see [Adding a Token Issuer](#).
- **Secure Sockets Layer (SSL) Certificates** are trusted certificates that you use to establish SSL communication with the external service. SSL Certificates are stored in the Trust Keystore.

CSF keys and certificates are configured from the **Settings > Credentials** dialog. CSF keys, certificates, and their values are specific to the AMCe instance where they are defined.

Managing Keys and Certificates

CSF keys and certificates are managed from the Credentials dialog.

As administrator, you manage the credential keys and certificates used by service developers from To open the Credentials dialog:

1. Click  and select **Settings > Credentials** from the side menu.
2. Click the **Keys, Certificates, or Token Issuer** tab.
3. Select an alias in the **Available Keys** or **Available Certificates** list to view the details of the key or certificate. Only the CSF keys and certificates that are currently in use are listed.
 - For CSF Keys, select **Show only referenced keys with null values** to see only keys that are referenced by artifacts that have no credential values.
 - For certificates, click **Export** to save the selected certificate to a file. You can then import the certificate for use in another instance.

Configuring a CSF Key

You can configure a new CSF key from the **Keys** tab in the Credentials dialog. You can edit the description, user name, or password of an existing key, but the key name can't be changed after it's created.

1. Click the **Keys** tab.
2. Click **Add** and provide the following values:
 - Unique key name. This name can't be changed after the key is created.
 - User name and password for the external system that requires this key for access.
3. Save the key.

Configuring a Web Service or Token Certificate

You can configure a new web service or token certificate from the **Certificates** tab in the Credentials dialog. You can't edit a certificate after it's created.

1. Click the **Certificates** tab.
2. Click **Add** and provide the following information:
 - **Alias** — Enter a unique name for the certificate.
 - **Content** — Copy the certificate definition into the text field. You can get Web Service certificate content from the system administrator of the service, or token certificate content from the remote identity provider.
3. Save the certificate.

 **Note:**

When a certificate is uploaded, it takes a few seconds before the certificate is available. Token certificates can take up to ten minutes.

To delete a certificate, click **X** by the selected alias in the list of **Available Certificates**. You can only delete certificates that you created.

Configuring an SSL Certificate

You can configure a new SSL Certificate from the **Certificates** tab in the Credentials dialog. You can't edit a certificate after you've created it.

1. Click the **Certificates** tab.
2. Click **Add** and provide the following information:
 - **Alias** — Enter a unique name for the certificate.
 - **Content** — Copy the certificate definition into the text field. You can get Web Service certificate content from the system administrator of the service, or token certificate content from the remote identity provider.
 - Select **Trusted SSL Certificate**.

3. Save the certificate.

 **Note:**

When a certificate is uploaded, it takes a few seconds before the certificate is available.

To delete a certificate, click **X** by the selected alias in the list of **Available Certificates**. You can only delete certificates that you created.

Disabling SSL Hostname Verification

Testing connectors can be difficult when they call an outbound service over SSL. If the SSL certificate has an incorrect or missing hostname, the developer might not be able to create the connector or might just have problems with testing.

You can make it easier to test a connector by turning off hostname verification for outbound SSL connections through the `Security_IgnoreHostnameVerification` policy.

 **Caution:**

Turning off hostname verification is a security risk. Setting this policy to `true` should be limited to development. When testing is complete, set the policy back to its default value of `false`.

This policy is set globally (`*.*.Security_IgnoreHostnameVerification`) and will affect all connectors. Setting the scope for a specific backend or connector is not supported.

For more information on configuring policies, see [Policies](#).

 **Note:**

Even if SSL hostname verification is disabled, you still need to import the SSL certificate if it's self-signed.

Adding a Token Issuer

To authenticate users with third-party tokens, you need to register the token issuers and associate them with their certificates.

After you've added at least one token certificate, use the steps below to add a token issuer from the **Credentials** dialog:

1. Click the **Token Issuers** tab.
2. Click **New Issuer**.
3. Enter the name of the token issuer in the **Name** field under **Issuer Details**.
4. Click **Add (+)** and select at least one name from the Select Certificate Subject Names dialog. All the certificates that have been uploaded are listed.

5. Save the token issuer.
6. If the list on the Token Issuers tab doesn't include your new issuer, click **Save** in the tab to update the list.
7. (Optional) Click **Rules** to configure a rule for a certificate subject name.

Configuring Rules

Rules govern how tokens provided by token issuers are processed. If the token provided by a token issuer doesn't meet the criteria specified by the rule, the request is rejected.

After you've added at least one token certificate and created a token issuer, you can configure a rule for the certificate subject name from the Token Issuer tab in the **Credentials** dialog.

1. On the Token Issuers tab, select a certificate subject name from the list.
2. Click **Rules**. As you add rules, the current number of rules is indicated on the Rules button.
3. Select **Enable Virtual User** if you're configuring rules for users that aren't registered.

With virtual users enabled, a token identifies a user with a record in IDCS, but roles are associated with the user based on the default content in the token, instead of on information in that account.

4. Under **Add a New Rule**, select the rule type.
5. Enter the required values for the rule type.
6. Click **Add**.

If you need to change a rule, just select it, make the updates and click **Done**. To delete a rule, select the rule and click **X**.

Rule Types

Filter Rule

The Filter rule consists of a token attribute and at least one value that must match the value associated with the token. The `name-id` attribute represents the username identified in the token, while the `user.tenant.name` attribute represents the tenant name associated with the token.

Use a comma-separated list to enter multiple attribute values for either attribute. If none of the values match, the token is deemed invalid. A value can contain a wildcard (*) character.

For example:

- `name-id=jack, jill, ann`
- `user.tenant.name=testing, development`

You can configure only one Filter rule per token attribute (that is, you configure one Filter rule with the `name-id` attribute and one Filter rule with the `user.tenant.name` attribute).

User Mapping Rule

The User Mapping rule defines how tokens are mapped to users, either by user name or email address. This rule is applicable only to JWT tokens, only if virtual users are disabled.

The rule consists of a token attribute, `name-id`, that represents the username identified in the token, and a user attribute name value of either `uid` or `mail`:

- `uid` is the user's username in the associated IDCS account (default behavior)
- `mail` is the user's email address in the IDCS account

You can configure only one User Mapping rule per issuer certificate name. If you don't configure a User Mapping rule, name matching is used (the default behavior).

Note:

For SAML tokens the User Mapping rule type is ignored and the default behavior is to map the username in the token to the username in the associated record.

Default Role Rule

The Default Role rule defines a list of roles to associate with users. This rule is applicable only if virtual users are enabled.

The rule consists of a list of role names that are assigned to all users presenting tokens verified using the corresponding token certificate. Use a comma-separated list to enter multiple attribute values.

For example: `role=technician, manager, tester`

You can configure only one Default Role rule per issuer certificate name. If you don't configure a Default Role rule, no roles are assigned to the requesting user unless you've configured a Role Attribute rule.

Role Attribute Rule

Use the Role Attribute rule to determine which roles to assign by examining the attributes in the token. If a Role Attribute role is defined, the token is searched for attributes with names that match any of the values defined in the rule. If matches are detected, the values of those token attributes are interpreted as roles and assigned to the virtual user. This rule is applicable only if virtual users are enabled.

The rule consists of a comma-separated list of token attribute names used to derive the roles that are assigned to users.

For example: `employeelevel, QAgrou`

You can configure only one Role Attribute rule per issuer certificate name, but you can use this rule in combination with the Role Mapping rule. If you don't configure a Role Attribute rule, no roles are assigned to the requesting user unless you've configured a Default Role rule.

 **Note:**

If you configure both the Default Role rule and the Role Attribute rule and the role attribute you defined is present in the token, the Default Role rule is ignored. However, if the defined role attribute isn't present, the roles specified in the Default Roles rule are applied to the virtual user. Role Mapping rules can also define which roles to use when no matches are found.

Role Mapping Rule

The Role Mapping rule associates roles with role attributes in the token identified by the Role Attribute rule. This rule is applicable only if virtual users are enabled.

The rule consists of an external role name, which is the value that should be found in one or more token attributes, and a list of roles to which the external role names are mapped. Use a comma-separated list to enter multiple attribute values.

For example: `employee=technician, manager, tester`

This example maps the external role name, `employee`, to the existing roles, `technician`, `manager`, `tester`.

 **Note:**

Role Mapping rules only work in conjunction with Role Attribute rules. If no Role Attribute rule is defined, Role Mapping rules are ignored. If the names of the token attributes configured in the Role Attributes rule don't match the external role names configured in the Role Mapping rule, the token attributes are treated as role names and are assigned to the requesting user. If the role names defined in the rule don't correspond to any existing roles, the value is ignored.

You can configure as many Role Mapping rules per issuer certificate as you need, but only one rule can be configured for each external role name. To map one external role to multiple roles, use a single rule and include all the roles in a comma-separated list, as shown in the example above.

Part II

Managing AMCe Artifact Lifecycles

This part includes the following chapters:

- [Understanding Lifecycles](#)
- [Client and App Profile Lifecycle](#)
- [Backend Lifecycle](#)
- [API Lifecycle](#)
- [API Implementation Lifecycle](#)
- [Connector Lifecycle](#)
- [Collection Lifecycle](#)

4

Understanding Lifecycles

Oracle Autonomous Mobile Cloud Enterprise provides a UI to simplify lifecycle management of your artifacts. As AMCe administrator, you use these features to create, maintain and publish backends, APIs and other artifacts.

An important part of your role as AMCe administrator is managing artifacts from implementation to production and maintaining them through multiple versions, also known as lifecycle.

The artifacts you'll be working with most often are clients, backends, custom APIs, connector APIs, and collections. In general, the same lifecycle phases apply to all of these artifacts.

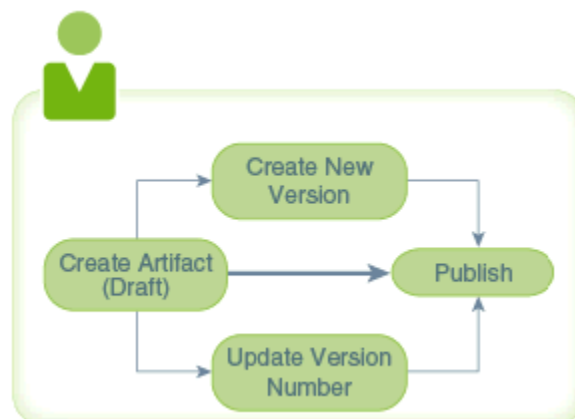
Throughout the development and testing phases of a project, artifacts can be created and edited in a **Draft** state, then **Published** (or moved to the trash). All artifacts are automatically assigned a version of 1.0 when they're created. During an artifact's lifecycle, new versions can be created and updated. Since artifacts are closely related to each other, it's important to keep track of dependencies and manage interactions.

This chapter introduces these important lifecycle phases and explains how you can work with an artifact through each of the phases and manage interactions between associated artifacts.

Draft State

When you create an artifact, whether it's a client, a collection, a custom API or any other type, the artifact has a **Draft** status. With a Draft version of an artifact, you can edit, create a new version, update an existing version, or remove the artifact (move it to the trash).

While an artifact is in the Draft state, you can experiment with it, modify it as many times as you need to, and test it thoroughly. You publish the artifact when you're satisfied with its configuration.



You can update the version number of an artifact in Draft state or create a new version to modify, then you can publish the updated artifact or the new version of the artifact.

Published State

When a specific version of an artifact is final, you can publish it. After it's published, that version of the artifact can no longer be edited. If you create a new version of an artifact that's in a Draft or Published state, the new version is created in the Draft state.

If there are no dependency issues, a published artifact can be exported to another instance. If you need to modify the artifact, you can create a new version of it and modify the new version. Because the new version is in the Draft state, you'll need to publish it before you can export it.

You can have multiple versions of an artifact. You can export different versions of an artifact to different instances.

If you've published an artifact by mistake or realize after it's been published that you need to make a change, you can create a new version of the artifact and make your changes to it. When you're satisfied with the configuration of the new version, you can publish it. For API implementations, AMCe automatically makes the latest version the default when the implementation is initially loaded. If the most recent version loaded isn't the implementation that you want associated with your API, you must explicitly specify a previously loaded implementation as the default.

You have the choice of keeping the previous version of an artifact as long as it's needed, or moving it to the trash (removing the artifact from the main view). In the case of a backend, you also have the option of changing its activation state to *inactive*. Artifacts in the trash are still accessible at runtime. For example, if you have a backend that calls `My_API` and someone moves `My_API` to the trash, the app can still call the backend and `My_API`.

All artifacts can be published independently and some can also be published when you publish their associated artifacts. For example, you can publish an API independently or the API can be published when you publish its associated backend.

When you publish an artifact, it's checked for any dependencies and whether or not those dependencies are already published. You'll be able to see the list of dependencies and can decide whether or not to proceed with publishing. If you decide to publish, any unpublished dependencies may be published too.

This table compares the Draft and Published states, behavior, and dependency considerations for backends, collections, custom APIs, and connector APIs:

Artifact Type	Permissible in Draft state	Permissible in Published state	Number of active versions per instance	Dependencies
Backend	Edit Create new version Update version Publish Export/import Manage activation Move to Trash	Create new version Manage activation Move to Trash	Multiple	Collection Custom APIs Connector APIs Roles
Collection	Edit Associate a backend Create new version Update version Publish Export/import Move to Trash	Create new version Move to Trash	Multiple	Roles
Custom API	Edit Create new version Update version Publish Export/import Move to Trash	Create new version Move to Trash	Multiple (Note: Only one API version per backend version)	Roles, Connector APIs, Custom APIs
Connector	Edit Create new version Update version Publish Export/import Move to Trash	Create new version Move to Trash	Multiple	None

Making Changes After a Backend is Published (Rerouting)

If you need to make backend fixes to your app, but the app's backend is already in production, there is a way that you can incorporate those changes into your app without having to recompile it — reroute the call to the backend.

Using a policy, you can reroute the call your app makes to the backend to a different backend that contains the needed fixes. First, publish the backend that contains the fix. Then, set the `Routing_RouteToBackend` policy, which lets you specify the original backend and redirect the call to the target backend with the fixes. Because your app is calling the original backend, there's no change to the `ClientID` or `ClientSecret`, which would require you to recompile the app binary.

Rerouting the call to a backend is useful when you want to make a minor fix that requires a change to the backend's metadata. Some instances where rerouting a published backend is useful:

- Making modifications to an API or a connector, such as adding an endpoint that you forgot.
- Changing the access permissions for an API.
- Changing the access permissions for a storage collection.
- Changing the offline sync property of a storage collection.
- Adding a storage collection to a backend, such as when you want to include a more efficient API implementation that needs storage for caching purposes.
- You have a change to the backend and you want to distribute the backend that has the fixes to other instances.

 **Note:**

The `Routing_RouteToBackend` policy should also be set when you're exporting or importing a package containing the target backend.

You can set `Routing_RouteToBackend` to specify that any API calls within the context of any version of the original backend are routed to the target backend:

- `OriginalBackend.*.Routing_RouteToBackend=TargetBackend(X.X)`
- `OriginalBackend(A.A).*.Routing_RouteToBackend=TargetBackend(X.X)`

For example: `FiF_Customer.*.Routing_RouteToBackend=FiF_Customer(3.2)`

Any API calls sent to any version of `FiF_Customer` are sent to `FiF_Customer, v3.2`.

 **Note:**

You can't use wildcards (*) in version values when setting the `Routing_RouteToBackend` policy.

You can also specify a particular version of the backend to route to a specific version of it. For example: `FiF_Customer(1.3).*.Routing_RouteToBackend=FiF_Customer(3.5)`

Any API calls sent to `FiF_Customer, v1.3` are sent to `FiF_Customer, v3.5`.

 **Note:**

If more than one redirect policy is defined for the backend, the policy defined with the fully-qualified backend takes precedence.

A call can be redirected to any backend, not just another version of the same backend. For example: `FiF_Customer(1.3).*.Routing_RouteToBackend=RepairIt(1.0)`

Any API calls to `FiF_Customer, v1.3` are sent to the backend `RepairIt, v1.0`.

You can also create a chain of rerouted calls to a backend. For example, a call to `backend_A` can be rerouted to `backend_B`. A second routing policy could redirect any calls to `backend_B` to `backend_C`. This would result in a call to `backend_A` being redirected to `backend_C`.

Packaging a Rerouted Backend

If you are exporting or importing a backend that is being rerouted, the Dependencies page includes a "Redirect to" statement that specifies the immediate target backend. Using the previous example, if a rerouting chain exists, and `backend_A` is being exported, the Dependencies page indicates a reroute to `backend_B`. Also, the `policies.properties` file lists only the routing policy for the backend in the package (`backend_A`).

Conditions for Rerouting a Backend

The following conditions apply whenever you reroute a backend:

- The original backend can be in an inactive state and be rerouted.
- If the app calls the original backend, notifications are sent and devices are registered using the client credentials associated with the original backend. However, if the app calls the target backend directly, then the clients from the target backend are used to send the notifications and register devices.
- If Social Identity is used to access an API and its associated backend is rerouted, the social authentication provider of the target backend should be selected and the access token from that provider should be entered in the Authentication section of the API Test page.
- If the original backend is exported, the target backend is not considered to be a dependency of the original.
- Generally, if either the original or target backend is included in an export or import package, the routing policy should be set when the export package is created or when the contents of the package are imported.
- When a backend is rerouted, the system log records the event. You can see which backends are being redirected from the log messages.

Versioning

Each time you create an artifact, it's assigned a version value of 1.0. As long as the artifact is in a Draft state, you can edit it, update its version, or create a new version (a major or minor incremental increase). As you develop your artifact, you can change the version's major and minor values (*Major.minor*).

When the major version number increments from 3.5 to 4.0 or even 6.0, it indicates that a significant change has been made to the current instance of the artifact, which likely affects its relationship to related artifacts. For example, changing the Web Services Description Language (WSDL) file of a SOAP Connector API would necessitate creating a new custom API implementation for it. A major change to a backend would also necessitate upgrading any mobile apps that use it.

After you create an artifact and it's still in a Draft state, you can change the version number with the **Update Version Number** command available from the landing page for your artifact type. For example, to update the version number for a backend, you

can go to the Backends page and select your backend, select **More > Update Version** and change the version number.

App Profile Versions

An app profile represents a specific version of a mobile app binary and the version number you assign to the app profile should correspond to the mobile app binary that it represents. When you create an app profile, you enter the version in the same format as the app binary. For example, if the app has a version of 3.1.2.3, the version you enter for the app profile is 3.1.2.3. No default version is applied to a new app profile.

While the app profile is in Draft state, you can update the version. For instance if the app version changes, you can change the version by opening the app profile and editing the **Version** field. After the app profile is published, you can create a new version through the **More > New Version** command on the App Profiles page.

Deleting an Artifact

Deleting an artifact permanently takes two steps. When you have an artifact that you don't need anymore, you can move it to the trash where it's kept until you're sure you want to delete. To delete an artifact permanently, you purge the artifact from the trash.

You can move an artifact that's in Draft or Published state to the trash. Depending on whether or not it's needed later, you can restore it or purge it.

Almost any artifact can be moved to the trash. For details on the conditions for removing artifacts, see [Moving an Artifact to the Trash](#).

When you restore an artifact, it retains the same state it had when it was moved to the trash. For details on restoration, see [Restoring an Artifact](#).

Purging is a permanent deletion and is available only from the Administration view. For details on how to permanently delete an artifact, see [Purging an Artifact](#).

Moving an Artifact to the Trash

Putting an artifact in the trash is considered a temporary deletion, the artifact is removed from the main view and is inaccessible to other artifacts. You can move an artifact that's in Draft or Published state to the trash. Depending on whether or not it's needed later, you can restore it or purge it.

You can't use artifacts that are in the trash because they can't be accessed, called, or executed. If you change your mind later or find you do need an artifact in the trash, you can restore it depending on the deletion policies.

The `Asset_AllowTrash` and `Asset_AllowUntrash` policies control the ability to move an artifact to the trash or restore it. You can set these policies to one of the following values:

- All
- None
- Draft
- Published

For details on setting policies, see [Policies](#).

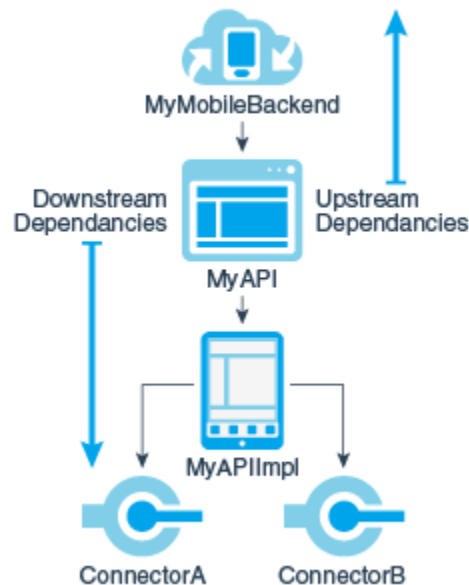
While it's in the trash, an artifact in Draft state can't be published, and any dependencies of that artifact can't be published regardless of whether or not those dependencies are in the trash. If you restore it, an artifact in Draft state can then be published.

 **Note:**

When an artifact is removed, its associated policies are removed along with the artifact and its dependencies. If the artifact is restored, the policies are also restored.

Upstream and Downstream Dependencies

You can move an artifact that's in Draft or Published state to the trash, but there are some conditions based on whether dependencies are involved. An artifact that's called by another artifact has an *upstream dependency*. An artifact that calls another artifact has a *downstream dependency*. If an artifact has dependencies that are active (not in the trash), you need to resolve the relationships to those dependencies before you can move the artifact to the trash.



For example, say you created an API called `MyAPI`. The backend that calls it, `MyBackend`, is the upstream dependency of `MyAPI`. The API calls its implementation, `MyAPIImpl`, is the downstream dependency of `MyAPI`.

Below are the common dependency scenarios that affect whether or not you can move an artifact to the trash, and whether or not dependencies of that artifact are also moved to the trash:

Case 1, Artifact is a dependency of a published artifact: If the artifact you want to remove is a dependency of a published artifact, you can't move the artifact in question to the trash because it would break its relationship with the published artifact. If you need to move the artifact to the trash, you must break the relationship between the artifacts first. For example, you want to move `MyAPI` to the trash but you can't because

it's a dependency of `MyBackend`, which is published. You have to break the relationship by moving `MyBackend` to the trash first, then moving `MyAPI` to the trash. If you need `MyBackend`, you can create a new version of it before moving the previous version to the trash.

Case 2, Artifact has tightly-coupled dependencies: If the artifact you want to remove has tightly coupled dependencies, moving it to the trash will remove the artifact *with its dependencies*. For example, if you need to remove an API that's associated with a real implementation or a connector API and its implementation, moving the API to the trash will also remove the implementation. (If the API is associated with a mock implementation, the relationship is broken and only the API is moved to the trash.)

Case 3, Artifact has dependencies: If the artifact that you want to move to the trash has dependencies that aren't tightly coupled, you must disassociate the artifact from its dependencies before you can move it to the trash. Only first-level upstream and downstream dependencies are considered. If there are any second-level dependencies (for example, the API's implementation calls a connector), you'll have to be aware of those relationships and resolve them prior to moving the artifact to the trash.

If the artifact has a dependency on a role, the artifact can be moved to the trash but not the role. Rule of thumb: Roles can't be trashed.

The following table lists the dependencies involved in a move to the trash. For each artifact type, it lists 1) the dependencies that are moved to the trash with the artifact, 2) the possible dependencies that may be associated with the artifact but are not moved to the trash with the artifact, 3) and any published upstream dependencies that would prevent a move to the trash.

Table 4-1 Dependencies Involved in a Move to the Trash

Artifact	Dependencies Moved to the Trash	Dependencies Not Moved to the Trash	Published Upstream Dependency That Prevents a Move to the Trash
Client	None	Backend App Profile	None
Backend	None	APIs Collections	Client
API	API Implementation Note: mock implementations can't be moved to Trash.	Backend API implementations that invoke the API Roles – Any role associated with the API is revoked. Roles can't be moved to Trash.	Backend
API Implementation	None	API that is implemented APIs that are called by the implementation	API that lists the implementation as active

Table 4-1 (Cont.) Dependencies Involved in a Move to the Trash

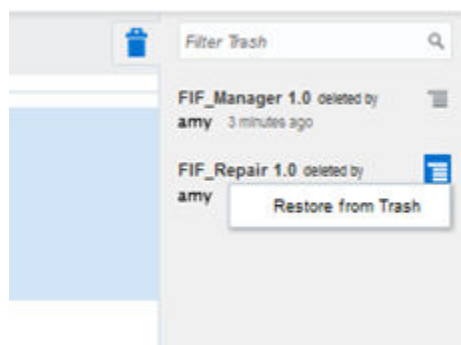
Artifact	Dependencies Moved to the Trash	Dependencies Not Moved to the Trash	Published Upstream Dependency That Prevents a Move to the Trash
Collection	None	Backend Roles – Any role associated with the collection is revoked. Roles can't be moved to Trash.	Backend
Connector	None	Backend API implementations that call the connector.	None

Restoring an Artifact

You might find that you need an artifact that's been moved to the trash. Restored artifacts retain the same state they had when they were moved to the trash. That is, an artifact in Draft state that was moved to the trash will still be in Draft state when restored.

As with moving an artifact to the trash, restoring an artifact has some considerations:

- If the artifact has no naming or version conflict, you can restore it by simply clicking the **Trash** (🗑️) and selecting **Restore from Trash** from the Trash drawer (☰) and confirming the restoration action.



- If duplicate artifacts exist (that is, artifacts with the same name and version) and one of these artifacts is in the trash, you can't restore the artifact. You must resolve the conflict first in one of the following ways and then restore the artifact:
 - Move the active artifact to the trash and restore the one already in the trash.
 - Change the version of the active artifact and then restore the one in the trash.


The following table lists the types of artifacts that can be restored and which dependencies are restored from the trash with each type of artifact. The last column lists the possible upstream and downstream dependencies of the artifact that are not in the trash and that could be affected by the restoration. These items are displayed in the Restore dialog as information only.

Artifact	Dependencies Restored With Artifact	Possible Artifact Dependencies Not in the Trash
Client	None	Backend
Backend	None	APIs Collections
API	Role	Backend API Implementation (non-mock) API implementation that calls the API
API Implementation	None	API that is implemented
Collection	Role	Backend
Connector	None	Backend

Detailed instructions for restoring an artifact in the trash are included for each artifact type in the chapters that follow.

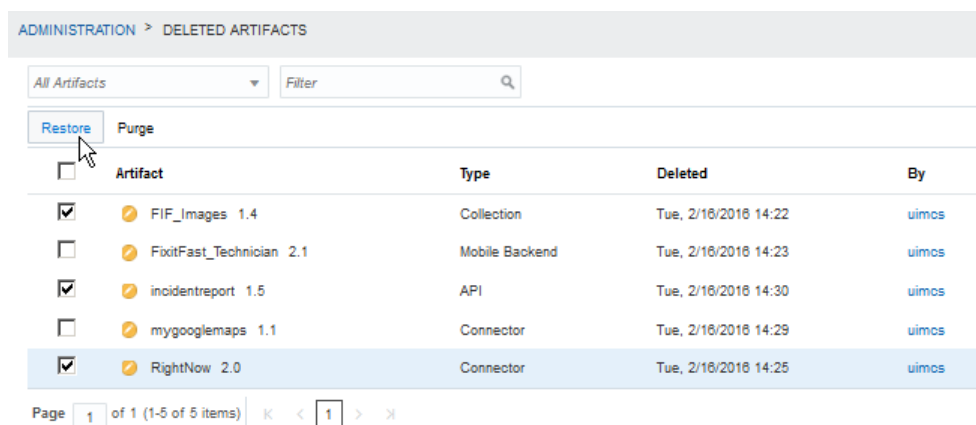
Restoring an Artifact from Administration

You can restore an artifact from the **Trash** menu as described above, or you can restore deleted artifacts from the Administration view.

1. Click  and select **Settings** from the side menu.
2. Click **Deleted Artifacts**.
3. Filter the list by selecting the type of artifacts you want to see. The default value is All Artifacts.

You can also use the Filter field to further refine the list:






- By the name of the artifact.
- By version number.
- By the name of the person who moved the artifact to the trash.



ADMINISTRATION > DELETED ARTIFACTS

All Artifacts Filter

Restore Purge

<input type="checkbox"/>	Artifact	Type	Deleted	By
<input checked="" type="checkbox"/>	 FIF_Images 1.4	Collection	Tue, 2/16/2016 14:22	uimcs
<input type="checkbox"/>	 FixitFast_Technician 2.1	Mobile Backend	Tue, 2/16/2016 14:23	uimcs
<input checked="" type="checkbox"/>	 incidentreport 1.5	API	Tue, 2/16/2016 14:30	uimcs
<input type="checkbox"/>	 mygooglemaps 1.1	Connector	Tue, 2/16/2016 14:29	uimcs
<input checked="" type="checkbox"/>	 RightNow 2.0	Connector	Tue, 2/16/2016 14:25	uimcs

Page 1 of 1 (1-5 of 5 items) < 1 >

4. Click the checkbox for each artifact you want to restore and click **Restore**.

To select all the items in the table at once, click the checkbox next to **Artifact** in the table header. Click again to clear all selections.

Artifact selection isn't persistent across pages. You can restore only the selected artifacts on the current page. If you want to restore artifacts listed across multiple pages, you'll have to restore the artifacts on the current page and then go to the next page.

Purging an Artifact

So how do you permanently delete an artifact? You must be a mobile cloud administrator and you purge it via the Deleted Artifacts tab from the Administration view. When an artifact is purged, it no longer appears in the list of trashed items and can't be restored.

Just as dependencies can affect restoring an artifact, they affect purging an artifact from the trash. If the artifact you want to purge has downstream dependencies, those dependencies are deleted along with the artifact. For example, when you purge an API in the trash, its implementation is deleted as well.


If the artifact is a downstream dependency of another artifact, you need to resolve the dependency with the other artifact before you can purge it.

The following table shows you which dependencies will be purged with a each type of artifact.

Artifact Type	Dependencies Purged with the Artifact	Dependencies Not Purged with the Artifact
Backend	None	APIs Collections
API	API implementations Roles	Backends Mock API Implementations
API Implementation	None	API implemented by the implementation
Connector	None	Backends
Collection	Roles	Backends API implementations that call the connector

Purging Artifacts from Administration

To permanently remove an artifact, you need to purge it from the trash. You can only purge artifacts from the Administration view.

1. Click  and select **Settings** from the side menu.
2. Click **Deleted Artifacts**.

By default, the list shows all artifacts in the trash. Artifacts are displayed in a descending order of when items were moved to the trash. You can change the display to list artifacts in alphabetical order of the person who moved the artifacts to the trash or by comments.

- Filter the list by selecting the type of artifacts you want to see. The default value is All Artifacts.

You can also use the Filter field to further refine the list:

- By the name of the artifact.
- By version number.
- By the name of the team member who moved the artifact to the trash.

You can also sort the order of the items in the trash by artifact, type, time the item was moved to the trash, or by the person who moved the item to the trash.

ADMINISTRATION > DELETED ARTIFACTS

All Artifacts Filter

Restore Purge

<input type="checkbox"/>	Artifact	Type	Deleted	By
<input checked="" type="checkbox"/>	FIF_Images 1.4	Collection	Tue, 2/16/2016 14:22	uimcs
<input type="checkbox"/>	FootFast_Technician 2.1	Mobile Backend	Tue, 2/16/2016 14:23	uimcs
<input checked="" type="checkbox"/>	incidentreport 1.5	API	Tue, 2/16/2016 14:30	uimcs
<input type="checkbox"/>	mygooglemaps 1.1	Connector	Tue, 2/16/2016 14:29	uimcs
<input checked="" type="checkbox"/>	RightNow 2.0	Connector	Tue, 2/16/2016 14:25	uimcs

Page 1 of 1 (1-5 of 5 items) < 1 >

- Click the checkbox of each artifact that you want to purge and click **Purge**.

To select all the items in the table at once, click the checkbox next to **Artifact** in the table header. Click again to clear all selections.

Artifact selection isn't persistent across pages. You can purge only the selected artifacts on the current page. If you want to purge artifacts listed across multiple pages, purge the artifacts on the current page and then go to the next page.

Artifact Lifecycles

Clients, backends, APIs, and other artifacts in AMCe each have an independent lifecycle.

In most respects, how an artifact is managed after it's created is the same regardless of whether it's a client, backend, collection, connector API, or a custom API. You've learned how to create an artifact, then modify it, and test it. Now that you have a viable artifact, it's time to publish it, perhaps create new versions or update existing versions and eventually deploy it to another environment for others to test and use.

There are some key details unique to different types of artifacts. The following chapters show you how to take each type of artifact through its lifecycle phases:

- [Client and App Profile Lifecycle](#)
- [Backend Lifecycle](#)
- [API Lifecycle](#)
- [API Implementation Lifecycle](#)

- Connector Lifecycle
- Collection Lifecycle

5

Client and App Profile Lifecycle


If your mobile app uses push notifications or you want to use analytics to examine and improve your app, you need a client. You associate the client, which represents a backend binary, with a backend. Use profiles to store notification credentials that can be shared between your clients. Clients go through similar lifecycle phases as other artifacts with a few differences, detailed in this chapter.

AMCe can help you manage client lifecycle. You can publish and export a client. You can modify its version number and move it to the trash when you don't need it anymore. Clients are top-level artifacts and their relationships with backends can affect how both types of artifacts are exported, imported, and moved to the trash.

If you want a general introduction to how artifacts interrelate in the overall lifecycle, see [Understanding Lifecycles](#).

Publishing a Client

When you're satisfied with a client configuration, you can publish it, but only if it's associated with a backend.

1. Click  and select **Mobile Apps > App Profiles** from the side menu.
2. Select the client that you want to publish.
3. Click **Publish**.

Dependencies are checked. If the associated backend is in Draft state, the confirmation dialog lists it and informs you that it will be published with the client. If the backend is already published, no dependencies are shown.

If the backend has downstream dependencies in Draft state, those dependencies will also be published. For example, `MyAppProfile 1.1` references `MyMobileBackend 1.0`. `MyMobileBackend` has dependencies on published `MyAPI2.2` and unpublished `MyAPI2.4`. When you publish `MyAppProfile 1.1`, the confirmation dialog only lists `MyMobileBackend1.0` as a dependency but `MyAPI2.4` is also published.


4. Click **Publish All**.

If the backend is in the trash, you won't be able to publish the client. Cancel the publish operation, and either restore the backend or associate the client with a different backend. Then try publishing again.

Usually, once an artifact is published it can't be changed. In the case of clients, you can add or remove the associated app profiles even if a client is published.

Updating the Version Number of a Client

When you create a client, you assign it a version number that is usually the version of the mobile app that the client represents. You can update its version number at any time if the client is in a Draft state. This is useful if a change to the binary was made and you need a new version designation.

1. Click  and select **Mobile Apps > App Profiles** from the side menu.
2. Open the client that you want to update from the list.
3. On the Settings page, change the value in the **Mobile App Version** field.

You'll get a message letting you know if you enter a duplicate version number.

Creating a New Version of a Client


You can create a new version of a client regardless of whether it's in a Draft or Published state. When you create a new version of a client, you're basically cloning the client configuration. You can then make changes to the new version. For example, although a client can be associated with only one instance of a backend, that backend can reference multiple clients. You could create new versions of a client, where each version corresponds to a specific platform of a mobile app (iOS, Android, and Windows), and then edit each client to reference the same backend. Another reason for creating new versions is to create multiple clients for the same platform if there are multiple mobile app binaries for the same platform that use the same backend.



Note:

Unlike other artifacts, which require that the version number use the *Major.minor* format, the version number for a client should be the same as the mobile app binary that's set by the app store. Depending on the version of the mobile app binary, the version could take the format of *Major.minor* or include an alphanumeric suffix with or without parentheses, a hyphen, space, or full stop. For example:

- 1.2
- 1.2 build 3452
- 1.2 (3452)
- 1.2-3452
- 1.2.3 (01-Jun-2016)

1. Click  and select **Mobile Apps > App Profiles** from the side menu.
2. Select the client and then select **More > New Version**.
3. Enter a version number in the **Mobile App Version** field. (The same as the mobile app binary set by the app store.)
4. Click **Save**.


The new version is created in a Draft state.

Moving a Client to the Trash

Remove a draft or published client by moving it to the trash. If it's needed later on, you can restore it from the trash.

 **Note:**



Moving a client to the trash does not move the associated backend or any profiles referenced by the client to the trash.

1. Click  and select **Mobile Apps > App Profiles** from the side menu.
2. Select the client, then select **More > Move to Trash**.
3. Click **Trash** in the confirmation dialog if there are no dependency issues.

If you think you or someone else might restore it later on, enter a brief comment about why you're putting this item in the trash.


To find out how dependencies can affect moving an artifact to the trash, see [Moving an Artifact to the Trash](#).

Restoring a Client

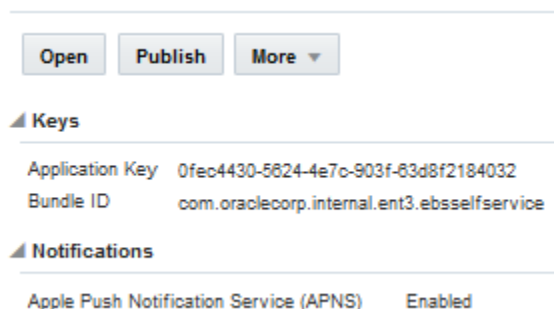
1. Click  and select **Mobile Apps > App Profiles** from the side menu.
2. Click Trash ()
3. In the list of items in the trash, select the client and select **Restore from Trash**.
4. Click **Restore** in the confirmation dialog if there are no conflicts.

Restoring an artifact can cause conflicts if a duplicate exists. To find out more about restoring an artifact when a duplicate artifact exists, see [Restoring an Artifact](#).


Managing Your Clients and App Profiles

When at least one client or app profile exists, you'll be taken to the Clients page every time you click  and select **Mobile Apps > App Profiles** from the side menu. On the left side of the page, you see a list of all the clients, their version numbers, and their Draft or Published state (clients in the trash aren't displayed).

On the Clients page, you can open, test, publish, and view see the dependencies and history for your clients:



- Click **Open** to see details about the selected client.

- Click **Publish** to change the state of the client.
- Click **More** to create a new version, export the client to another instance, or move the client to the trash.
- Click **Trash** () to see which clients are in the trash.
- Expand **Keys** to obtain the values for the client ID and the application key.
- Expand **Notifications** to see which push notifications, if any, are enabled for this app profile.

On the lower right side of the page, you can view data about the selected client:

- Expand **Dependencies** to see the backend and app profile that this client references.

 **Note:**

If the backend has downstream dependencies, go to **Mobile Apps > Backends** and view them from the Dependencies section of the selected backend.

- Expand **History** to quickly see the latest activity for the client.

Click the **Profiles** tab to view available app profiles and create new app profiles for your clients.

6

Backend Lifecycle

You created a backend and now it's time to use it. Remember that after you publish it, it becomes immutable, that is, you can't modify it.

If you want to make a change to a published backend, you create a new version of it. Because backends are tightly integrated with custom code, APIs, and other objects in AMCe, you'll need to consider the relationships and dependencies on those objects.

If you want a general introduction to how artifacts interrelate in the overall lifecycle before exploring the lifecycle of backends, see [Understanding Lifecycles](#).

Backend Lifecycle States

Relationships with other artifacts create dependencies. For example, your backend might depend on other artifacts, such as collections or APIs. When any artifact changes state, all dependent artifacts must also change states. AMCe keeps track of any dependencies for you.

Backends have the following activation states that determine whether they can be updated, deleted, or whether or not a new version can be created:

- **Active:** Denotes the version of the backend is valid and active.
- **Quiesce:** Denotes the version of the backend has become quiet, that is, it no longer supports new requests, and after all currently running requests are completed, it's changed to Inactive. This is a transitional state.
- **Inactive:** Denotes the version of the backend that's present but not in an active state (that is, not usable).

If a user tries to access an API through an inactive backend, a 404 code is returned.

- **Deleted:** Denotes the version of the backend that's been moved to the trash and susceptible to a hard delete (actually removed from the repository).




Note:

Only mobile cloud administrators can purge (that is, permanently delete) an item in the trash.

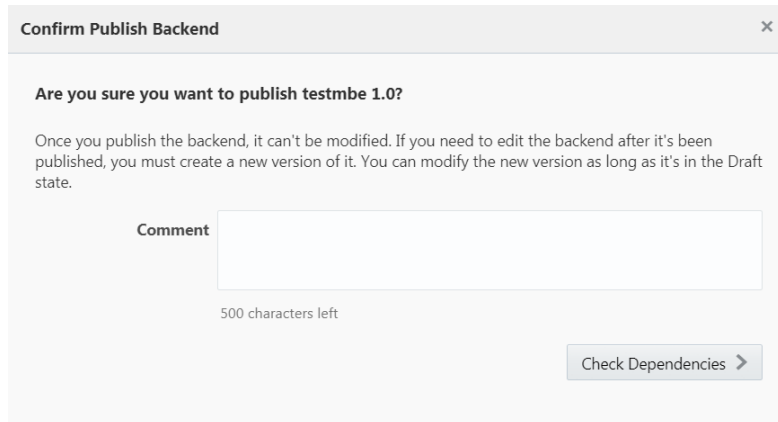
Publishing a Backend

Follow these steps to publish a backend. When a backend is published, all dependencies that aren't yet published must also be published.

1. Click  and select **Mobile Apps > Backends** from the side menu.
2. Select the backend that you want to publish.

3. Click **Publish**.


The Confirm Publish dialog opens:



4. In the Confirm Publish Backend dialog, click **Check Dependencies** to reveal whether or not the backend has dependencies and what those dependencies are so you'll know how to proceed:
- If you don't have dependencies, a confirmation dialog is displayed. Click **Publish**.
 - If any dependencies are found in the trash, they're listed. Cancel the publish operation, restore the dependent items from the trash, and restart the process.
 - If there are dependencies in the Draft state, they're listed in the confirmation dialog. You have the option to publish all the dependent artifacts along with your mobile backend. Click **Publish All**.

Updating the Version Number of a Backend

If you created a new version of a backend using the New Version dialog, you can update its version number if it's still in a Draft state. This is useful if you need to designate a different version number for it before you publish it or you've made a change to the configuration and you need a new version designation.

1. Click  and select **Mobile Apps > Backends** from the side menu.
2. Select the backend you want to update from the list.
3. In the right section, select **More > Update Version Number**.
4. Enter a version number of the format *Major.minor*.

The previous version of the backend is displayed next to the field. You'll get a message letting you know if you've entered an existing version number.


5. (Optional) Add a brief description that states what distinguishes this version from the previous one.
6. Click **Update**.

A confirmation message is displayed. A draft of the new version is added to the list of backends.

Creating a New Version of a Backend

When you create a new backend, the version is automatically set to 1.0. As long as the backend is in a Draft state, you can change any aspect of it. As you develop your backend, you can change the version's major and minor version values as you see fit.

You can use a published backend as a root for a new version.

1. Click  and select **Mobile Apps > Backends** from the side menu.
2. Select the published backend.
3. In the right section, select **More > New Version**.

The new version is created in a Draft state.

Note:

If the backend is associated with an API, you can't associate another version of that API with the backend, regardless of whether the backend is in a Draft or Published state. You must create a new version of the backend and associate it with the other API version.


Moving a Backend to the Trash

Remove a backend in a by moving it to the trash. A backend in the trash is no longer listed but it's still viable, that is, it could continue to serve requests. If the backend is needed later on, you can restore it from the trash.

Note:

If a backend is referenced by a client, you can't move that backend to the trash. If the backend is in Draft state, you can disassociate it from the client by opening the backend, selecting **Clients** in the navbar and clicking **Delete (X)** for that client. Then you can move the backend to the trash.

An alternative to removing a backend is to deactivate it, in which case it no longer services requests. See [Deactivating a Mobile Backend](#) for information.

1. Click  and select **Mobile Apps > Backends** from the side menu.
2. Select the backend.
3. In the right section, select **More > Move to Trash**.
4. Click **Trash** in the confirmation dialog if there are no dependency issues.


If you think you or someone else might restore it later on, enter a brief comment about why you're putting this item in the trash.

To find out how dependencies can affect moving an artifact to the trash, see [Moving an Artifact to the Trash](#). To restore a backend that's in the trash, see [Restoring a Backend](#).

If you move a backend to the trash that has been redirected to another backend, the redirection still occurs.

Deactivating a Backend

If you want to stop access to a backend without deleting it, you can do so by deactivating it. A deactivated backend can't service any more requests. Deactivation is most common for backends in a Published state that have been replaced by newer versions and are no longer needed.

1. Click  and select **Mobile Apps > Backends** from the side menu.
2. Select your backend and click **More > Manage Activation**.
3. In the dialog that appears, select **Inactive** from the drop-down list to deactivate the backend, or **Active** to reactivate an inactive backend.
4. Click **Save**.


If you deactivate a backend that has been redirected to another backend, the redirection still occurs.

Restoring a Backend

1. Click  and select **Mobile Apps > Backends** from the side menu.
2. Click **Trash** (.
3. In the list of items in the trash, click  by the backend you want and select **Restore from Trash**.
4. Click **Restore** in the confirmation dialog if there are no conflicts.

Restoring an artifact can cause conflicts if a duplicate exists. To find out more about restoring an artifact when a duplicate artifact exists, see [Restoring an Artifact](#).


Managing a Backend

When at least one backend exists, you'll be taken to the Backends page every time you click  and select **Mobile Apps > Backends** from the side menu. On the left side of the page, you see a list of all the backends, their version numbers, and their Draft or Published state (mobile backends in Trash aren't displayed).

On the upper right side of the Backends page, you can open, test, publish, see runtime data about your backend, and get authentication and application key values:

Open Publish More ▾

Metrics



Normal

0 API Calls
Last 10 minutes


0.0 Avg Response
seconds


Keys ?

OAUTH CONSUMER KEY		Refresh	Revoke
Client ID	768b6f89-3b80-49ea-ab26-add27ecefaf6f		
Client Secret	Show		

HTTP BASIC AUTHENTICATION		Refresh	Revoke
Mobile Backend ID	26247a51-be61-4286-955c-7f73b4f71276		
Anonymous Key	Show		

Client Applications Manage

NAME	APPLICATION KEY
 MyClient	081cbfba-2d21-4b58-9c12-4b8bc646aee8

- Click **Trash**  to see which backends are in the trash.
- Click **Open** to see details about the selected backend.
- Click **Publish** to change the state of the backend from Draft to Published.
- Click **More** to create a new version, update an existing version, change the activation state, or move the backend to the trash.
- Look in the **Metrics** section to see the number of calls to the API associated with the backend and the average response time.
- Expand **Keys** to obtain the values for the backend ID, anonymous key (click **Show**), and the application key for the associated client. Click **Manage** to configure the associated clients or create a new client.

On the lower right side of the page, you view data about the selected backend:

- Expand **Dependencies** to see the artifacts the mobile backend is dependent on.
- Expand **Used By** to see any associated artifacts that aren't dependencies.
- Expand **History** to quickly see the latest activity for the selected backend.

7

API Lifecycle

The lifecycle stages of custom APIs and API implementations are similar. Both artifacts go through a design-time phase where each is created, tested, edited, and then published.

When you create a new custom API, its version is automatically set to 1.0 and it's considered to be in a Draft state. You can test and edit your draft API as often as needed. As you develop your API, you can change the version's major and minor values as you see fit, that is, creating a new version of your API or updating an existing version.

After you've implemented and tested your API, and you're satisfied with your API configuration, you can publish it with the understanding that a published API can't be changed. To make a change to a published API, create a new version of the API. APIs are implemented with custom code. For custom APIs, you'll also need to create a new implementation for the new version.

You can export a draft or published API for use in other instances. Eventually, the API may become obsolete, and you can move it to the trash.

If you want a general introduction to how artifacts interrelate in the overall lifecycle before exploring the lifecycle of APIs, see [Understanding Lifecycles](#).

Custom APIs and API Implementations

AMCe tracks a custom API as it's created, saved, published, implemented, deactivated, and reactivated. Custom APIs can be published independently or when a related backend is published.

Scope and Version Format

Both custom APIs and API implementations have versions that use the format `Major.minor`.

Active Versions

Though there can be multiple active versions of an API implementation, a single implementation is mapped to a specific API version.

Draft and Published States

Both custom APIs and API implementation can have a Draft state or a Published state.

- A custom API can be published independently or published when a related backend is published.
- An API implementation can be published independently.

Actions Tracked by AMCe

The following operations are tracked for custom APIs: Create, Update, Publish, and Move to Trash.

The following operations are tracked for API implementations: Create and Save.

Backend References

A backend can reference multiple APIs, but a backend can only reference one version of a specific API. For example, a backend can reference both `myAPI1.1` and `yourAPI2.0`, but it can't reference both `myAPI1.1` and `myAPI2.0`.

API implementations aren't referenced directly by a backend. The implementation is referenced by the API, which is in turn referenced by the backend.

Dependencies

A custom API is dependent on its active API implementation (as determined by the policy). In other words, an API implementation is a dependency of any APIs that list it as the active or default implementation.

An API implementation is dependent on the API that it implements and any other APIs called by the custom code (as listed in the file manifest).

Backends are also dependencies of associated custom APIs.

Policy Attributes

A custom API is affected by the API version to implementation policy mapping and the default API version setting.

An API implementation is affected by the number of node instances per virtual machine and standard runtime policies such as read-only, log-levels, etc.


For detailed descriptions of policies and their values, see [Oracle Autonomous Mobile Cloud Enterprise Policies](#).

Publishing a Custom API

As soon as an API is published, it can't be changed. You can create a new version of it, but you cannot edit it.

Note:

You must have an implementation associated with the API to publish it. A mock implementation is provided by default. To associate an implementation other than the mock implementation, open the API, and click **Implementations** in the left navigation bar. Select the implementation you want and click **Set as Default**.

1. Click  and select **Applications > APIs** from the side menu.
2. Select the draft API that you want to publish.

3. Click **Publish**.

You can enter a justification for publishing in the Comment field.


When the API is published, you're returned to the APIs page where you can see the updated status of your API.

 **Note:**

Custom APIs can be published independently of implementations. When you publish an API, the implementation isn't published automatically. To understand the relationship between custom APIs and their implementations, see [Custom APIs and API Implementations](#).

Updating the Version Number of an API

If you created a new version of an API using the New Version dialog, you can update the version number of the API if it's still in a Draft state. This is particularly useful if you need to designate a different version number for it before you publish the API.

1. Click  and select **Mobile Apps > APIs** from the side menu.
2. Select the API you want.
3. Select **More > Update Version Number**.
4. Enter a version number of the format *Major.minor*.

The previous version of the API is displayed next to the field. You'll get a message letting you know if you enter an existing version number.

5. (Optional) Add a brief description that states what distinguishes this version from the previous one.
6. Click **Update**.

A confirmation message is displayed. A draft of the new version is added to the list of APIs.

Creating a New Version of an API

You can make a new version of a custom API regardless of whether it's in a Draft or Published state. When you create a new version of a custom API, you are basically cloning the API configuration and making changes to it alone. You can specify the implementation to associate with the new version of the API. You can upgrade your custom API easily by creating a new version of it:

1. Click  and select **Mobile Apps > APIs** from the side menu.
2. Select the API.

You can create a new version of a custom API whether it's in a Draft or Published state.


3. In the right section, select **More > New Version**.

AMCe checks for any dependencies on other APIs and for an associated implementation.

4. Enter a version number in the format *Major.minor*.
If you enter a version number that already exists, you'll get a message letting you know that number is already in use.
5. (Optional) Add a brief description that states what distinguishes this version from the previous one.
6. Click **Create**.
A confirmation message is displayed. A draft of the new version is created and is visible in the API Catalog.




Moving a Custom API to the Trash

Remove a custom API by moving it to the trash. If the API is needed later, you can restore it from the trash.

1. Click  and select **Mobile Apps > APIs** from the side menu.
2. Select the custom API you want to remove.
3. In the right section, select **More > Move to Trash**.
4. Click **Trash** in the confirmation dialog if there are no dependency issues.
If you think you or someone else might restore it later on, enter a brief comment about why you're putting this item in the trash.

To find out how dependencies can affect moving an artifact to the trash, see [Moving an API Implementation to the Trash](#).

Restoring a Custom API


1. Click  and select **Mobile Apps > APIs** from the side menu.
2. Click **Trash** (.
3. Make sure *APIs* is selected in the trash drawer.
4. In the list of items in the trash, click  by the API you want and select **Restore from Trash**.
5. Click **Restore** in the confirmation dialog if there are no conflicts.

When you restore an API, its implementations are not restored with it. You'll have to manually restore the implementations you want and designate an implementation as the default. Open the restored API, click **Implementations** from the navbar, and set an implementation as the default.

Restoring an artifact can cause conflicts if a duplicate artifact already exists. To restore an artifact when a duplicate artifact exists, see [Restoring an Artifact](#).

Managing an API

After you create a custom API, you'll want to edit it, publish it, see what implementations are associated with it, in short, you want to be able to manage the API and examine details of the APIs created by other service developers. The APIs page gives you access to all these features.

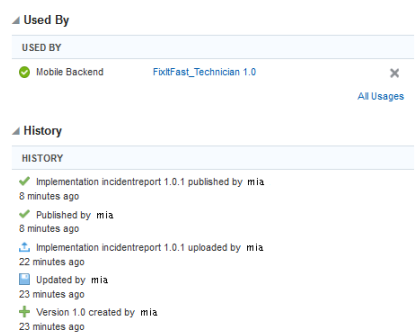
When at least one custom API exists, you'll be taken to the APIs page every time you click  and select **Mobile Apps > APIs** from the side menu. On the left side of the page, you'll see a list of all the custom APIs except for those in the trash. You can see which APIs are in the Draft state and which are in the Published state. Every API is listed by its name and version number.

The right side of the page is where you can open, test, publish, and examine data about your custom API.

On the upper right side of the APIs page, you can perform the following actions:

- Click **Open** to view details and settings for the selected custom API.
- Click **More** to create a new version, update an existing version, or move an API to the trash.
- Expand **Implementations** to see what implementations are available, along with their version numbers and whether they are in a Draft or Published state. Click **Manage** to go directly to the Implementations page.

On the lower right side of the page, you view data about the selected API:



- Expand **Used By** to see the list of the backends that call on the API. Click **All Usages** to see the complete list.
- Expand the **History** section to quickly see the latest activity for the selected custom API.

8

API Implementation Lifecycle


After you have an API implementation in a Draft state that's configured and tested, you're ready to publish it. API implementations go through the same lifecycle phases as APIs, in addition to being published, new versions can be created, existing versions can be updated, and obsolete implementations can be moved to the trash.

Remember that after an API implementation is published, it can't be changed. If you're still configuring and testing the implementation, keep it in a Draft state until it's ready for the next phase of the lifecycle.

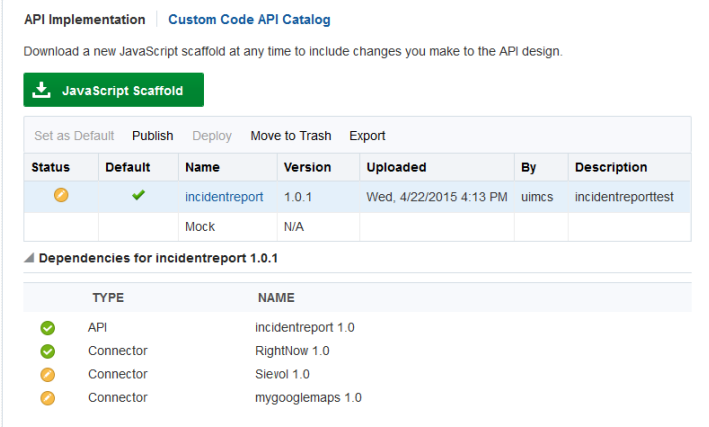
If you want a general introduction to how artifacts interrelate in the overall lifecycle before exploring the lifecycle of API implementations, see [Understanding Lifecycles](#).

Publishing an API Implementation



You can publish an implementation that contains real, non-mock data from the API Implementation page. Implementations can be published independently of APIs and can have separate versions as well. This lets you make changes to a published implementation, such as minor modifications or bug fixes, without requiring the API itself to be updated.





1. Click  and select **Mobile Apps > APIs** from the side menu.
2. Select the API associated with the implementation that you want to publish.
3. Expand **Implementations** in the right section and click **Manage**.

The API Implementation page is displayed:



The screenshot shows the 'API Implementation' page for 'Custom Code API Catalog'. It includes a 'JavaScript Scaffold' button and a table of implementations. Below the table is a section for 'Dependencies for incidentreport 1.0.1'.

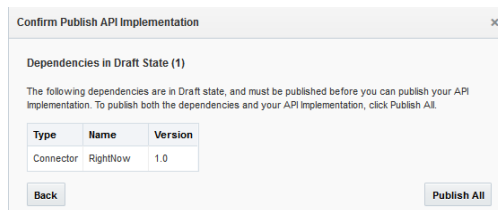
Status	Default	Name	Version	Uploaded	By	Description
		incidentreport	1.0.1	Wed, 4/22/2015 4:13 PM	uimcs	incidentreporttest
		Mock	N/A			

TYPE	NAME
 API	incidentreport 1.0
 Connector	RightNow 1.0
 Connector	Sievel 1.0
 Connector	mygooglemaps 1.0

You can see the list of dependencies by expanding the **Dependencies** section of the API Implementation page. The API associated with the implementation and any other APIs or connectors that the implementation calls are listed. You can see which dependency is in a Draft state, a Published state, or is unresolved.

4. Select the implementation and click **Publish**.

A dependency search is performed. If unresolved dependencies are found, the implementation can't be published. Resolve the issue and try publishing the implementation again.



If any API dependencies are declared through the `Oracle-Mobile-API-Version` header instead of through the `package.json` file, the API Designer isn't aware of dependencies declared through the header and won't prompt you with information when you publish the calling API. In this case, you must remember to publish the dependent API yourself.

5. If unpublished dependencies are found, click **Publish All** to publish all the listed unpublished artifacts.

If you don't want to publish all the dependencies with your implementation, click **X** to cancel the operation. You can either publish the dependencies individually or edit your implementation to remove them.

Creating a New Version or Updating the Version of an API Implementation

Implementations can be published independently of APIs and can have separate versions as well. This lets you make changes to a published implementation, such as minor modifications or bug fixes, without requiring the API itself to be updated. You can create a new version of an API Implementation that is in a Draft or Published state. If you want to make changes to a published implementation, you must create a new version of it.

If you have previously uploaded an implementation with a given version specified and that implementation is still in a Draft state, you can replace that version without incrementing the version number. This might be desirable if you've uploaded the implementation and find, after testing the implementation, that there are further changes that you need to make before you can publish the changes. After you've published a version, that version is final.

You can also update the version number of an implementation in a Draft state. The process for both is the same. You set the `version` attribute in the implementation's `package.json` file.


1. Open the `package.json` file and change the `version` attribute. For example, change `"version": "1.0"` to `"version": "1.1"`.
2. Upload a zip file of the modified implementation to the associated API version.

Some key points to know about implementation versions are:

- Implementation versions are maintained independently of API versions. When you publish an API, the implementation isn't published automatically.
- When you upload a new version of an implementation, it becomes the default version (active implementation) for that API. You can change the default version in the API's Implementations page.
- The custom API's `Routing_BindApiToImpl` policy defines the association between an API version and the implementation version.

Moving an API Implementation to the Trash

Remove an API implementation by moving it to the trash. If the implementation is needed later, you can restore it from the trash.

1. Click  and select **Mobile Apps > APIs** from the side menu.
2. Select the API associated with the implementation.
3. Click **Implementations** in the API navigation bar.
4. Select the draft API implementation to remove.
5. Click **Move to Trash**.




Only real implementations (not mock implementations) can be moved to the trash. If you're moving the current default implementation to the trash, the next most recent version of the implementation is automatically set to the default. If no other implementations exist, the mock implementation is made the default.

6. Click **Trash** in the confirmation dialog if there are no dependency issues.

If you think you or someone else might restore it later on, enter a brief comment about why you're putting this item in the trash.

To find out how dependencies can affect moving an artifact to the trash, see [Moving an Artifact to the Trash](#). To restore an API implementation that's in the trash, see [Restoring an API Implementation](#).

Restoring an API Implementation

1. Click  and select **Mobile Apps > APIs** from the side menu.
2. Select the API associated with the implementation.
3. Click **Trash** (.
4. Select **Implementations** in the trash drawer.
5. In the list of items in the trash, click  by the implementation you want and select **Restore from Trash**.
6. Click **Restore** in the confirmation dialog if there are no conflicts.

If you're restoring an implementation that was used by an API, the implementation won't be restored as the default (active) implementation for the API. You'll have to reset the implementation as the default from the Implementations page (select the API and click **Implementations** in the navbar).

Restoring an artifact can cause conflicts if a duplicate artifact already exists. To restore an artifact when a duplicate artifact exists, see [Restoring an Artifact](#).

9

Connector Lifecycle

The lifecycle stages of all connectors are the same. Each type of connector goes through a design-time phase where each is created, tested, edited, and then published.


For all connectors, there are the creation phase, the testing and editing phase, and the publishing phase. When you create a new connector, its version is automatically set to 1.0 and it's considered to be in a Draft state. In the Draft phase, you can test and edit your API as often as needed. When you're satisfied with your connector configuration, publish it with the understanding that a published connector can't be changed.

As you develop your connector, you can change the version's major and minor values as you see fit, that is, creating a new version of your API or updating an existing version. After you've implemented and tested your connector, you can publish it. Eventually, a connector may become obsolete, and you can move it to the trash.

If you want a general introduction to how artifacts interrelate in the overall lifecycle before exploring the lifecycle of connectors, see [Understanding Lifecycles](#).

Publishing a Connector

Before you can use a connector, you need to publish it:


1. Click  and select **Mobile Apps > Connectors**.
2. Select the draft connector that you want to publish.
3. Click **Publish**.

(Optional) You can enter a justification for publishing the connector in the **Comment** field.

When the connector API is published, you're returned to the Connectors page where you can see the updated status of your connector.

Updating the Version Number of a Connector

If you created a new version of a connector using the New Version dialog, you can update the version number of the connector if it's still in a Draft state. This is particularly useful if you want to create an alternate version of the current connector or need to designate a different version number before you publish the connector.

1. Click  and select **Mobile Apps > Connectors** from the side menu.
2. Select the connector from the list.
3. In the right section, select **More > Update Version Number**.
4. Enter a version number of the format *Major.minor*.


The previous version of the connector is displayed next to the field. You'll get a message letting you know if you've entered an existing version number.

5. (Optional) Add a brief description that states what distinguishes this version from the previous one.
6. Click **Update**.

A confirmation message is displayed. A draft of the new version is added to the list of connectors.

Creating a New Version of a Connector

You can make a new version of a connector regardless of whether it's in a Draft or Published state. When you create a new version of a connector, you're basically cloning the connector configuration and making changes to it. You can make minor changes or expand upon already defined functionality. A major update can result in a disruption of mobile services to your customers due to invalid values being requested or returned, an inability to read the same file formats as the previous version, and so on.

1. Click  and select **Mobile Apps > Connectors** from the side menu.
2. Select a connector from the list.

You can create a new version of a connector whether it is in a Draft or Published state

3. In the right panel, select **More > New Version**.
4. Enter a version number in the format *Major.minor*.


If you enter a version number that already exists, you'll get a message letting you know that number is already in use.

5. (Optional) Add a brief description that states what distinguishes this version from the previous one.
6. Click **Create**.

A confirmation message is displayed. A draft of the new version is added to the Connector page.

Moving a Connector to the Trash

Remove a connector by moving it to the trash. If the connector is needed later, you can restore it from the trash.


1. Click  and select **Mobile Apps > Connectors** from the side menu.
2. Select the connector.
3. In the right section, select **More > Move to Trash**.
4. Click **Trash** in the confirmation dialog if there are no dependency issues.

If you think you or someone else might restore it later on, enter a brief comment about why you're putting this item in the trash.

To find out how dependencies can affect moving an artifact to the trash, see [Moving an Artifact to the Trash](#).

To restore a connector that's in the trash, see [Restoring a Connector](#).


Restoring a Connector

1. Click  and select **Mobile Apps > Connectors** from the side menu.
2. Click **Trash** ()
3. In the list of items in Trash, click  by the connector you want and select **Restore from Trash**.
4. Click **Restore** in the confirmation dialog if there are no conflicts.

Restoring an artifact can cause conflicts if a duplicate artifact already exists. To restore an artifact when a duplicate artifact exists, see [Restoring an Artifact](#).

Managing a Connector

After you create a connector, you'll want to edit it, publish it, see what artifacts are associated with it, in short, you want to be able to manage the connector and examine details of the connectors created by other service developers. The Connectors page gives you access to all these features.

When at least one connector exists, you'll be taken to the Connectors page every time you click  and select **Mobile Apps > Connectors** from the side menu. On the left side of the page, you see a list of all the connectors except for those in the trash. You can see which connectors are in the Draft or Published state. Every connector is listed by its name and version number.

The right side of the Connectors page is where you can open, test, publish, or examine data about the connector:

On the right side of the page, you can perform the following actions:

- Click **Open** to see details about the selected connector.
- Click **More** to create a new version, update an existing version, or move an connector to the trash.
- Expand **Used By** to see the list of the implementations that call on the connector.
- Expand **History** to quickly see the latest activity for the connector.

10

Collection Lifecycle


The collection lifecycle involves moving from the Draft state to the Published state.

After you publish a collection, it can't be modified. While you can publish a collection and also create a new version of a collection, you can also remove a collection as described in [Moving a Collection to the Trash](#).

If you want a general introduction to how artifacts interrelate in the overall lifecycle before exploring the lifecycle of collections, see [Understanding Lifecycles](#).

Publishing a Collection

You create a collection within the context of a backend. When you're satisfied with that collection, you can publish it.

1. Click  and select **Mobile Apps > Storage** from the side menu.
2. Select the collection you want to publish.
3. In the Details section on the right, click **Publish**.

Note:

You can publish a draft collection whenever you feel that it's complete. After it's published, however, it can't be changed.

A collection can also be published *involuntarily* when a backend associated with a collection is published. If the associated collection isn't yet published, it will be published automatically to support the backend.

When a collection is published:


- Its metadata (its description and access roles) are frozen. To update the collections metadata, you must create a new version.
- The major version (given the version that you arbitrarily defined) is incremented.
- It's no longer in your personal development space. It's available for anyone with the proper permissions to associate with a backend.
- Instance data isn't moved with the collection.

Note:

Instance data (such as actual user objects or actual collection objects stored in collections) is typically created at runtime, or by user scripts or code as part of a configuration. It isn't moved with the collection.

Updating the Version Number of a Collection

When you update a version, the new number is backward-compatible and the collection history continues.

1. Click  and select **Mobile Apps > Storage** from the side menu.
2. Select a collection.
3. In the Details section, select **More > Update Version Number**.
4. Specify an optional comment and click **Update**.

The collection history reflects the incremented number.

Creating a New Version of a Collection

You can't copy a collection, but you can save yourself some time by creating a new version of an existing collection. If you create a new version, you'll have the same data. It is possible to rename the collection and reset the version as long as the collection is in a Draft state.


If you want a collection that starts with 1.0 and that has the same data as another collection, you must make a new collection and import the data.

When you create a new version number, an independent collection is spawned from that point with a new history that's unrelated to the previous collection. Any data is carried forward to the new version.




Note:

A collection can't have more than one version of an object.

1. Click  and select **Mobile Apps > Storage** from the side menu.
2. Select a collection.
3. In the Details section on the right, select **More > New Version**.
4. Specify an optional comment and click **Update**.

Moving a Collection to the Trash

Remove a collection by moving it to the trash. Moving a collection to the trash means it's no longer listed but it's still viable. If the collection is needed later, you can restore it.

1. Click  and select **Mobile Apps > Storage** from the side menu.
2. Select the collection you want to remove.
3. In the Details section on the right, select **More > Move to Trash**.
4. Click **Trash** in the confirmation dialog if there are no dependency issues.


If you think you or someone else might restore it later on, enter a brief comment about why you're putting this item in the trash.

Although only a mobile cloud administrator can purge a collection (eliminate it permanently), you can delete an object in a collection using the command-line operation, `DELETE`.

To find out how dependencies can affect moving an artifact to the trash, see [Moving an Artifact to the Trash](#).

To restore a collection in the trash, see [Restoring a Collection](#).


Restoring a Collection

1. Click  and select **Mobile Apps > Storage** from the side menu.
2. Click **Trash** (.
3. In the list of items in the trash, click  by the collection you want and select **Restore from Trash**.
4. Click **Restore** in the confirmation dialog if there are no conflicts.

Restoring an artifact can cause conflicts if a duplicate artifact already exists. To restore an artifact when a duplicate artifact exists, see [Restoring an Artifact](#).

Managing a Collection



After you create a collection, you'll want to edit it, publish it, and in short, manage the collection and examine details of collections created by other mobile developers. The Storage page gives you access to all these features.



When at least one collection exists, you'll be taken to the Storage page every time you click  and select **Mobile Apps > Storage** from the side menu. On the left side of the page, you'll see a list of all the collections except for those in the trash. You can see which collections are in the Draft state and which are in the Published state. Every collection is listed by its name and version number.

The upper right side of the page is where you can open, test and publish the selected collection:

- Click **Open** to see details about the selected collection.
- Click **Publish** to change the state of your collection from Draft to Published.
- Click **More** to create a new version, update an existing version, associate the collection with a backend, or move a collection to the trash.

On the lower right of the page, you can examine usage and history details:

▲ Used By			
	Mobile Backend	FixItFast_Technician 1.1	×
	Mobile Backend	FixItFast_Technician 1.0	×

▲ History			
	Published by mia	Last Friday at 12:10 AM	
	Updated by mia	Last Thursday at 11:41 PM	
	Created version 1.0 by mia	Last Thursday at 11:40 PM	

- Expand **Used By** to see which backends are associated with the collection. To disassociate the selected collection from an artifact that uses it, click **X** next to the artifact's name.
- Expand the **History** section to quickly see the latest activity for the selected collection.

Part III

Reference

- [Oracle Autonomous Mobile Cloud Enterprise Policies](#)

A

Oracle Autonomous Mobile Cloud Enterprise Policies

This chapter lists the policies that you can configure in Oracle Autonomous Mobile Cloud Enterprise (AMCe). Policies control a variety of things, including logging level, password expiration times, means for restricting user access, and proxies. Policies can affect all artifacts of a specific type, or they can affect an individual artifact.

Note:

The scope value shown is the narrowest level at which the property can be set.

AMCe Policies and Values

Policies determine the behavior of various aspects of AMCe. As AMCe administrator, you can view and modify the policies in the `policies.properties` file by exporting the file from the Administration page.

Policy	Type / Description	Default Value	Scope / Affects
<code>Analytics_ApplicationGuid</code>	String. Stores an association between the backend and the Analytics application. The value is the Application ID.	There is no default value for this policy.	Scope: Backend Affects: Backend
<code>Analytics_BaiduMapCsfKey</code>	String. Stores the name of the CSF key that stores the Baidu application key (ak).	There is no default value for this policy.	Scope: Backend Affects: Backend
<code>Asset_AllowPurge</code>	String. Controls whether or not Draft and Published artifacts in the trash can be purged (deleted permanently). Valid values are: <ul style="list-style-type: none">• All• None• Draft• Published	All	Scope: Instance Affects: Backend, Custom API, API Implementation, Connector, and Collection

Policy	Type / Description	Default Value	Scope / Affects
Asset_AllowTrash	String. Controls whether or not Draft and Published artifacts can be moved to the trash. Valid values are: <ul style="list-style-type: none"> • All • None • Draft • Published 	All	Scope: Instance Affects: Backend, Custom API, API Implementation, Connector, and Collection
Asset_AllowUntrash	String. Controls whether or not Draft and Published artifacts can be restored from the trash. Valid values are: <ul style="list-style-type: none"> • All • None • Draft • Published 	All	Scope: Instance Affects: Backend, Custom API, API Implementation, Connector, and Collection
Asset_DefaultInitialVersion	String. Sets the default version for all newly created artifacts.	1.0 Note: Generally, the default value should be used.	Scope: Instance Affects: All artifacts that have versions
CCC_DefaultNodeConfiguration	String. Sets the default node.js configuration used by the API implementation (custom code). The custom code implementation can override the default node configuration in its package.json.	For instances of AMCe provisioned before 18.3.3, the default node.js configuration by default is 6.10. For instances provisioned as 18.3.3 or above, the default node.js configuration by default is 8.9.4.	Scope: Instance Affects: Custom Code

Policy	Type / Description	Default Value	Scope / Affects
	<p>Valid values are:</p> <ul style="list-style-type: none"> 8.9 - the service uses node.js 8.9.4 and the following libraries: "agentkeepalive": "3.3.0", "bluebird": "3.5.1", "body-parser": "1.18.2", "express": "4.16.2", "http-proxy-agent": "2.0.0", "https-proxy-agent": "2.1.0", "method-override": "2.3.10", "request": "2.83.0" 6.10 - the service uses node.js 6.10.10 and the following libraries: "agentkeepalive": "3.1.0", "bluebird": "3.4.6", "body-parser": "1.15.2", "express": "4.14.0", "http-proxy-agent": "1.0.0", "https-proxy-agent": "1.0.0", "method-override": "2.3.6", "request": "2.74.0" 0.10 - the service uses node.js 0.10.25 and the following libraries: "agentkeepalive": "0.2.2", "bluebird": "2.9.30", "express": "3.5.1", "http-proxy-agent": "0.2.6", "https-proxy-agent": "0.3.5", "method-override": "2.2.0", "path": "0.4.9", "request": "2.34.0", "url": "0.7.9" 		
CCC_LogBody	<p>Boolean. Determines whether to log the body of a request in custom code. Bodies will be logged in the following circumstances:</p> <ul style="list-style-type: none"> Logging level == FINEST or there is an uncaught exception. This property is set to true. 	false	<p>Scope: Backend Affects: Custom Code</p>
CCC_LogBodyMaxLength	<p>Integer. Sets the maximum number of characters to log if the custom code is logging the request body.</p>	512	<p>Scope: Backend Affects: Custom Code</p>
CCC_SendStackTraceWithError	<p>Boolean. Determines whether or not to send the stack trace from node.js with the REST response from the custom code container indicating that there is a code problem.</p>	false	<p>Scope: Backend Affects: Custom Code</p>
Connectors_Endpoint	<p>String. Stores the endpoint URL of the particular connector instance. Set this policy by uncommenting the policy.</p>	<p>There is no default value for this policy. The initial value is set when the connector is created.</p>	<p>Scope: Connector Affects: Connectors</p>
Connector_Ics_Connections	<p>String. Identifies the JSON document representing connections to each configured ICS instance.</p>	null	<p>Scope: Instance Affects: ICS Connector</p>

Policy	Type / Description	Default Value	Scope / Affects
Database_CreateTablesPolicy	String. Controls whether the Database API can create, alter, or drop tables from custom code or SQL. The default value (<code>allow</code>) enables calls from custom code that perform implicit operations and also explicit query operations from raw SQL. Setting this policy to <code>implicitOnly</code> enables these operations and JSON from custom code calls, and prohibits SQL operations. Setting the policy to <code>explicitOnly</code> enables these operations using the Database Management Service API, and prohibits non-SQL operations from custom code. Setting the policy to <code>none</code> curtails implicit and explicit table creation, deletion, and updates.	allow	Scope: Instance Affects: Database Service
Database_MaxRows	Integer. Sets the maximum number of rows that can be returned by a single database query.	1000	Scope: Instance Affects: Database Service
Database_QueryTimeout	Integer. Sets the number of seconds to wait for a database query to return before canceling it.	20	Scope: Instance Affects: Database Service
Diagnostics_ExcludedHttpHeadersInLogs	String. Creates a list of headers that shouldn't be logged with each API request in the API History log file.	Authorization header, cookie name	Scope: Instance Affects: Administration
Diagnostics_RequestPercentageErrorThreshold	Double. Sets the percentage of requests returning error codes compared with total request above which the service will report an error condition. Set this value higher than the one set for the <code>Diagnostics_RequestPercentageWarningThreshold</code> policy, which sets the adverse level of system health.	10	Scope: Instance Affects: Administration

Policy	Type / Description	Default Value	Scope / Affects
Diagnostics_RequestPercentageWarningThreshold	Double. Sets the percentage of requests returning error codes compared with total request above which the service will report a warning condition.	1	Scope: Instance Affects: Administration
Logging_Level	Integer. Sets the logging level.	800	Scope: Backend Affects: Custom APIs, Storage
Network_HttpConnectTimeout	Integer. Sets the amount of time spent in milliseconds (ms) connecting to the remote URL. The value should be less than the value of Network_HttpRequestTimeout.	There is no default value for this policy. The initial value is set when the connector is created.	Scope: Instance, Backend, Connector, Fully-Qualified Connector Affects: Connectors
Network_HttpReadTimeout	Integer. Sets the maximum time (in milliseconds) spent waiting to read data. The value should be less than the value of Network_HttpRequestTimeout.	There is no default value for this policy. The initial value is set when the connector is created.	Scope: Instance, Backend, Connector, Fully-Qualified Connector Affects: Connectors
Network_HttpRequestTimeout	Integer. Sets the amount of time in milliseconds (ms) on an HTTP request before it times out.	40,000 ms	Scope: Instance Affects: Custom APIs
Notifications_DeviceCountWarningThreshold	Double. Defines the threshold level (percentage) of messages sent successfully without returning an error. If the proportion of messages accepted by the service provider is below the threshold, then a warning is displayed. The default value is 70.0 (70%). Set this policy as needed.	70.0 Note: For testing purposes only, consider setting this value to 100.0 (100%).	Scope: Instance Affects: Notifications
Routing_BindAPIToImpl	String. Determines which core service to use to resolve the API request.	There is no default value for this policy.	Scope: API Affects: Custom APIs, Connectors
Routing_BindAPIToMock	Boolean. Resolves the API request to a mock service instead of the implementation that's bound to the API.	false Note: Do not modify this policy.	Scope: Fully-Qualified API Affects: Backend, Custom APIs

Policy	Type / Description	Default Value	Scope / Affects
Routing_DefaultImplementation	String. Specifies the default implementation for the initially created API (that is, the mock service).	MockService/1.0 Note: Do not modify this policy.	Scope: Instance Affects: Custom APIs
Routing_RouteToBackend	String. Reroutes mobile API calls made to a backend to the target backend specified.	There is no default value for this policy.	Scope: Backend Affects: Dispatcher
Security_AllowOrigin	String. Enables Cross Origin Resource Sharing (CORS) from HTML5 clients on an external domain. Supported values are: <ul style="list-style-type: none"> disallow url1, url2, url3 - specifies a whitelist of URLs from which cross-site requests to APIs can be made. If the origin of the cross-site request matches one of the patterns in the whitelist, the request is allowed. Otherwise, access is restricted. The wildcard character, *, can be used when providing URL values but doesn't apply across dot (.), forward slash (/), or colon (:) characters.	disallow Note: When dealing with browser-based applications, it's highly recommended that cross-site access to APIs either be restricted completely, or be restricted to trusted origins where legitimate applications are known to be hosted to prevent vulnerability to cross-site attacks (e.g., Cross-Site Request Forgery).	Scope: Instance Affects: All cross origin calls to a given instance
Security_AuthTokenConfiguration	JSON Object. Provides a configuration to integrate with third-party identity providers that support JWT, which mobile app users can use to authenticate.		Scope: Environment Affects: Security

Policy	Type / Description	Default Value	Scope / Affects
Security_CollectionsAnonymousAccess	<p>A comma-separated list of storage collections following this pattern:</p> <pre><collection1_name>[(<version> *)] [, <collection2_name>[(<version> *)]] [, ...]</pre> <p>Sets a storage collection to allow anonymous access. For each storage collection listed in the policy, anonymous read and write access will be allowed, provided that the correct anonymous access key is defined in the request headers. Specifying '*' as the version allows anonymous access to all versions of the collection.</p>	No default value	<p>Scope: Storage Collections</p> <p>Affects: Only the listed Collections</p>
Security_ExposeHeaders	<p>String. Provides a means for browsers to access the server whitelist headers. By default, Cross Origin Resource Sharing (CORS) disallows accessing returned headers by the browser.</p> <p>Applies to HTML5 clients accessing a given resource from an external domain.</p>	<p>""</p> <p>Indicates that no response headers are to be exposed to the browser.</p>	<p>Scope: Instance</p> <p>Affects: All cross origin calls to a given instance</p>
Security_IdentityProviders	<p>String. Stores identity providers configuration.</p>	<p>Facebook identity provider configuration</p>	<p>Scope: Instance</p> <p>Affects: Security</p>
Security_IgnoreHostnameVerification	<p>Boolean. Disables the SSL host name verification.</p> <p>To be applied to connectors (in development) that call outbound services using SSL certificates with an invalid or incomplete hostname.</p>	false	<p>Scope: Instance</p> <p>Affects: REST, SOAP, ICS, and Fusion Applications Connectors</p>
Security_OwsmPolicy	<p>Object. Sets the security policy used for outbound security.</p>	<p>There is no default value for this policy.</p> <p>The initial value is set when the connector is created.</p>	<p>Scope: Connector</p> <p>Affects: Connectors</p>

Policy	Type / Description	Default Value	Scope / Affects
Security_SsoRedirectWhitelist	String. Lists the URL patterns for the SSO redirect_uri parameter values that are permitted.	disallow	Scope: Instance, Backend Affects: SSO Token Relay
Security_TokenExchangeTimeoutPolicy	String. Defines the policy that governs the expiration time for tokens generated and issued as a result of token exchange. Valid values are: <ul style="list-style-type: none"> FromTimeoutSecs - token expiry time is governed by the Security_TokenExchangeTimeoutSecs policy. FromExternalToken - token expiry time is set to the same time as the external token expiry time. FromExternalTokenLimitedByTimeoutSecs - token expiry time is set to the value determined from the Security_TokenExchangeTimeoutSecs policy or the external token expiry time, whichever comes first. 	FromTimeoutSecs	Scope: Instance Affects: SSO Token Exchange
Security_TokenExchangeTimeoutSecs	Integer. Sets the token expiration time for SSO login.	216000 s	Scope: Instance Affects: SSO Token Relay

Policy	Type / Description	Default Value	Scope / Affects
Security_TransportSecurityProtocols	<p>String. Specifies a list of the TLS/SSL protocols that should be used for the outbound connection for the specific connector. By default, only TLSv1.2 protocols are used for outbound connections. This property can be used to override the system defaults so that connections can be established to legacy systems that don't support new versions of TLS/SSL.</p> <p>Caution: Use this property carefully as older protocols are more vulnerable to security exploits.</p> <p>Valid value is a comma separated list of the TLS/SSL protocols. Note that extra spaces around the protocol names are ignored. For example, TLSv1, TLSv1.1, TLSv1.2.</p> <p>Supported protocols are: SSLv2Hello, TLSv1, TLSv1.1, TLSv1.2.</p>	No default value	<p>Scope: Connectors, Fully-qualified Connectors</p> <p>Affects: All Connectors</p>
Sync_CollectionTimeToLive	<p>Integer. Sets the default amount of time that data requested by a mobile app from a storage collection remains in the local cache that's used by the Synchronization library.</p>	86400 s Set this policy as needed.	<p>Scope: Instance</p> <p>Affects: Storage</p>
Url_PercentEncodeQueryParameterSpaces	<p>Boolean. Controls how spaces in query parameters of a URL are encoded. If set to true encodes spaces as %20; and encodes them as + otherwise. Spaces in other parts of the URL are always encoded as %20.</p>	false	<p>Scope: Connector</p> <p>Affects: REST Connector</p>