
Jitsuin Archivist

Release v0.1.0

jitsuin.com

Dec 06, 2022

CONTENTS:

1	Introduction	1
1.1	LifeCycle	1
1.2	Behaviours	5
1.3	Attachments	5
1.4	Locations	6
1.5	Access Policies	7
1.6	Compliance Policies	10
2	Administrator configuration	11
2.1	Register your Azure Active Directory with Jitsui	11
2.2	Assign Users for Interactive Use	12
2.3	Configure Client Credentials for Non-Interactive Access	13
2.4	Getting Access Tokens (client secret)	17
2.5	Testing Access	17
2.6	Using PingOne for Enterprise as IDP	20
3	API Request Authorization and Authentication	29
4	Tenancies API	31
4.1	Tenancy Information	31
5	TLS CA Certificates Management (v1)	33
5.1	TLS CA Certificates Upload (v1)	33
5.2	TLS CA Certificate Retrieval (v1)	34
5.3	TLS CA Certificate Deletion (v1)	36
5.4	TLS CA Certificates Update (v1)	36
5.5	TLS CA Certificates Swagger API	37
6	Compliance API	43
6.1	Compliance	43
6.2	Creating Compliance Policies	43
6.3	Types of Compliance Policies	44
6.4	Compliance Policy Creation	47
6.5	Compliance Checking	47
6.6	Compliance Swagger API	48
7	Identity and Access Management (v1)	55
7.1	IAM Access Policies API (v1)	55
7.2	IAM Subjects API (v1)	74
8	Attachments API	83

8.1	Retrieve Attachment	83
8.2	Attachments Swagger API	84
9	Attachments V2 API	87
9.1	Retrieve Attachment	87
10	BlobsV1 API	89
10.1	Upload Blob	89
10.2	Retrieve Blob	90
11	Locations API	91
11.1	Location Creation	91
11.2	Location Retrieval	92
11.3	Locations Swagger API	94
12	Assets API	101
12.1	Asset Record Creation	101
12.2	Asset Record Retrieval	103
12.3	Attachments Operations API	106
12.4	Builtin Operations API	108
12.5	RecordEvidence Operations API	113
12.6	Assets Swagger API	114
13	Events API	119
13.1	Event Record Creation	119
13.2	Event Record Retrieval	119
13.3	Events Swagger API	122
14	Blockchain API (v1alpha2)	127
14.1	Blockchain Retrieval (v1alpha2)	127
14.2	Blockchain Swagger API	128
15	System API	131
15.1	Archivistnode	131
15.2	Archivistnode Swagger API	133
16	Miscellaneous	137
16.1	API Request Paging	137
	HTTP Routing Table	143

INTRODUCTION

1.1 LifeCycle

Jitsuin Archivist is an asset lifecycle assurance system built to lower business risk. It enables enterprises to reveal, reduce and report risks introduced through connected devices and Digital Transformation. Jitsuin Archivist maintains a Full Service History for assets and Internet Things by keeping a permanent shared record of *When Who Did What to a Thing*. Following the principle that you can't secure what you can't see, this level of visibility and traceability is key to lowering the risk of business transformation with the Internet of Things.

Jitsuin Archivist makes recording and auditing the full lifecycle of an asset simple: any authorized participant (including an agent or endpoint on the device itself) can register the events they know about, from which all participants can see a relevant aggregate picture of the asset's maintenance and operational history. By knowing *When Who Did What to a Thing* human actors and connected systems can make stronger judgments about the trustworthiness of a device and the data it produces.

1.1.1 Security Twins and Trustworthiness

Trust is subjective. Compliance policies are a judgement call. No matter what security technology you have in play every trust decision you make will depend on the circumstances: who is accessing what; where they're coming from; how sensitive an operation they're attempting; the consequences of getting it wrong. An asset that is safe in one context may not be in another.

Difficult decisions are slow to be made, and slow decisions means lost time. No matter what security mechanisms are in place, if you can't *trust* the assets and data in your systems then the chances are you won't be able to automate decisions and processes, keeping humans in the loop and losing the scale advantages of AI and IoT.

Security Twins take the heavy lifting out of this process by providing a simple trust layer that any client or process can ask. They answer the question: given all that I know about this asset - its security status, maintenance history, active alarms - can I trust it with what I'm trying to do right now? They help you make smarter, faster, better decisions *and* provide traceability that enable you to explain those decisions later on.

1.1.2 Security Twins (Jitsuin Archivist Asset Records) vs Digital Twins

A Digital Twin simplifies Digital Transformation by representing real-world objects in virtual form that allows business applications to connect to Internet Thing data through simple APIs. This solution avoids stressing the power-constrained devices and unreliable network connectivity that often go hand-in-hand with IoT installations.

Security Twins - built from Asset Records in Jitsuin Archivist - are not Digital Twins, although they do share a number of useful characteristics. Security Twins provide a trust layer that underpins the trustworthiness of Digital Twin applications. Understanding the differences between the two will help you to get the best out of the Archivist system and your Digital Transformation.

Digital Twin	Security Twin
Typically connected to an agent on the physical twin via real-time interfaces	Doesn't rely on a direct connection to the physical twin. Any authenticated actor may report on asset state
Typically seeks to model sensor state and 'hot data' from the device	Concentrates on operational and maintenance state of the device. Sensor data is typically only reported if it is out of safe range or compliance
Acts as a proxy for collecting any reading from the device when a direct connection is not feasible	Acts as a Full Service History for the device and its maintenance operations
Acts as an endpoint for business applications to gather data or receive commands	Provides the necessary information and insight from the asset history and status to augment Digital Twin data with a score of trustworthiness, provenance and compliance

Security Twins and Digital Twins work well together to provide a full picture of Internet of Things operations: the Digital Twin offering a reliable and efficient view of the most recent known state of the device sensors, and the Security Twin (Archivist) providing an immutable record of its operational history and firmware state at the times of those readings. The APIs in this SDK can be used to integrate Jitsuin Archivist with an IoT platform such as Microsoft Azure IoT Hub, thereby bringing the power of a Security Twin layer without disrupting telemetry and data operations.

1.1.3 Asset Record Creation

The first event in an Asset Record's life is *creation*. This initializes its Full Service History record and begins tracking the device to which that record pertains. Devices must be registered in the Archivist system through the creation of an Asset Record before any other lifecycle events can be recorded and handled.

Tracked vs Untracked assets

Because the Full Service History of devices lasts forever it is not possible to actually delete Asset Records from the system. Instead there is a concept of *tracked* assets (those that are interesting to the system and actively recording lifecycle events), and *untracked* assets (those that are no longer actively interesting, for example as a result of decommissioning).

This state is reflected in the `tracked` attribute. When an Asset Record is created its `tracked` attribute is automatically set to `TRACKED`.

When a real asset is disposed of for any reason it may be desirable to remove it from the Jitsuin Archivist system so that it does not appear in lists or searches. Assets can be removed from default lists and searches by setting the `tracked` property to `UNTRACKED`, but they will still be present in the system with their Service Histories available.

1.1.4 Asset modification

Asset attributes can be modified after creation by using the API. Only the asset identity is fixed: all other attributes can be modified.

Modification of significant attributes (such as firmware version) results in an audited entry in the asset history.

1.1.5 Recording events in the asset lifecycle

Any interaction with a device can be significant: from user logins to unexpected restarts or ad-hoc observations. Keeping a record of these events can build up a picture of how an asset came to be in its current state and provide crucial insight to future maintenance staff, auditors, and security remediation teams.

Archivist allows recording these general lifecycle events through the `RecordEvidence` behaviour.

For more detail, see [RecordEvidence Operations API](#)

1.1.6 Firmware security management

One of the most common sources of security breach in connected systems is unpatched software in devices. By providing a common platform for all relevant participants to collaborate on device maintenance, Jitsuin Archivist simplifies effective vulnerability disclosure, device recalls, regulatory compliance, and improved decision making.

As well as recording the current firmware version of each device in its Asset Record, a full history of prior versions, vulnerability warnings, patches, and times left unpatched is maintained.

Handling the firmware lifecycle record of an asset is performed through the `Firmware` behaviour. Devices can be marked as requiring an update for any reason using the `updateRequired` operation, are marked as vulnerable through the `vulnerability` operation, and record successful patching through the `update` operation.

To aid in analysis and compliance management these events can be linked by adding a common tag to the pair of messages in the `arc_correlation_value` attribute.

For more detail, see `firmware_behaviour`

1.1.7 Maintenance management

There are many factors that affect the security and trustworthiness of a device beyond firmware bugs: expired digital IDs, access control failures, and physical tampering among them. Jitsuin Archivist records and manages these events through the `maintenanceRequest` and `maintenance` event types.

To aid in analysis and compliance management these events can be linked by adding a common tag to the pair of messages: for example, a work order number.

For more detail, see `maintenance_behaviour`

1.1.8 How to create an Asset Record

Creating an Asset Record with the APIs is fast and straightforward. Follow the steps in this order for best success:

1. Determine if the asset is associated with a particular location. If it is, look up the location identity (if it already exists) or get a new location identity by creating one.
2. Import any attachments associated with the asset (for example, photograph of the physical unit) and get their identities.
3. Create the Asset, including the (optional) identities of the location and attachments.

For more detail, see [Location Creation](#), [Upload Blob](#) and [Asset Record Creation](#)

1.1.9 Understanding asset lifecycle events

For details of how to retrieve Asset Records from the system see [Asset Record Retrieval](#).

Lifecycle events in the Archivist system give stakeholders a shared view of “When Who did What to a Thing”. The “What” and the “Thing” are quite straightforward, but the “When” and “Who” can be more nuanced.

Timestamps

Once committed to the Jitsuin Archivist system, each lifecycle event record carries 3 separate timestamps:

- `timestamp_declared` - an optional user-supplied value that tells when an event happened. This is useful for cases where the client system is off-line for a period but the user still wishes to record the accurate time and order of activities (eg inspection rounds in an air-gapped facility). If unspecified, the system sets `timestamp_declared` equal to `timestamp_accepted` (see below).
- `timestamp_accepted` - the time the event was actually received on the Jitsuin Archivist node’s REST interface. Set by the system, cannot be changed by the client.
- `timestamp_committed` - the time the event was confirmed distributed to all DLT nodes in the value chain. Set by the system, cannot be changed by the client.

Having these 3 fields enables users of Jitsuin Archivist to accurately reflect what is claimed, whilst also preventing tampering and backdating of entries.

User principals

Once committed to the Jitsuin Archivist system, each lifecycle event record carries 2 separate user identities:

- `principal_declared` - an optional user-supplied value that tells who performed an event. This is useful for cases where the user principal/ credential used to connect to the Archivist system does not accurately or usefully reflect the real-world agent (eg a multi-user application with device-based credentials).
- `principal_accepted` - the actual user principal information belonging to the credential used to access the Jitsuin Archivist node’s REST interface. Set by the system and retrieved from the authorizing IDP, cannot be changed by the client.

1.2 Behaviours

Jitsuin Archivist supports a wide range of cyber-physical assets with very different capabilities and properties: some will have simple firmware maintenance lifecycles; others may have complex software configurations to manage; and yet others may have significant physical maintenance requirements to meet.

‘Behaviours’ are the technical mechanism through which Asset Records (and by extension estate security and compliance posture) are managed. Behaviours represent aspects of the asset lifecycle such as firmware updates or physical maintenance.

Within each behaviour there is a set of ‘operations’ which update the asset’s service history and create the basis for the shared evidence base for compliance insights, patch performance and so on.

1.2.1 Declaring allowed Behaviours

In order to maintain integrity of assets’ service histories and the overall security and compliance posture of an estate it is necessary to restrict which behaviours are pertinent to each asset, and therefore what statements can be made about the asset’s lifecycle. It is therefore necessary to supply a list of allowed behaviours when first creating the Asset Record.

1.2.2 System-defined attributes

Certain asset attributes are manipulated and interpreted by the system in order to provide compliance and security insights for the asset. In order to get the best benefit from the Archivist it is essential to take care to handle these properties carefully and consistently when passing them to behaviours operations.

All asset properties with an `arc_` prefix are used for system-defined purposes. Most operations also support custom properties which can have arbitrary names, though it is recommended to adopt a naming scheme which minimizes chances of collisions or confusion with other clients in the value chain.

1.3 Attachments

Attachments in Jitsuin Archivist enable images, PDFs and other binary data to be attached to assets and events. This brings added richness to the evidence base and facilitates high fidelity collaboration between stakeholders.

1.3.1 Uploading an Attachment

The first step is to upload the attachment to the attachments API and noting the uuid of the uploaded attachment. See *Upload Blob*

1.3.2 Attachments on assets

Adding an attachment to an asset enables recording of characteristics of the asset that are very difficult to capture in text. For example, the specific placement of a component or the device’s serial number or rating plate.

While asset attachments are generally expected to be unique, the same attachment can be applied to multiple assets, such as a stock image of a type of device.

See *Attachments Attach*

The asset 'arc_primary_image'

Attachments to assets are named in their `arc_display_name` property, so that they can be searched and indexed. Names are arbitrary and may be defined according to the needs of the application, but one name is reserved and interpreted by the Jitsuin stack: `arc_primary_image`.

If an asset has an attachment named `arc_primary_image` then this will be used by the user interface and other Jitsuin tools to represent the asset.

1.3.3 Attachments on events

Adding an attachment to an event allows recording and communication of asset status that is difficult to capture in text. For example:

- a photograph of physical state of a device such as alignment of components or wear on tamper seals at the time of a particular inspection.
- a PDF of a safety conformance report to support a maintenance event.
- a software manifest to support an update.
- an x-ray image

To add attachments to an event simply specify an `attachments` list in the `attributes` of the request block when posting an event. See [Attachments Attach](#) for an example of the JSON layout.

1.4 Locations

Assets in Jitsuin Archivist are arranged into *locations*, which allows logical assets (eg digital twins) to be grouped together in a physical context (eg a single plant location).

This enables users of the system to quickly identify the answers to questions such as “*how many PLCs in the Greyslake plant need to be updated?*”, or “*who was the last person to touch any device in the Cape Town facility?*”. It is not *required* for assets to be associated with a location, but it is a useful way to group assets in the same physical location without inventing your own link values in custom attributes.

It is important to recognize that the location does not necessarily denote the asset’s current position in space: it simply determines which facility the asset belongs to. For things that move around, GIS location information can be included with any event in the Asset Record, and the asset’s `arc_last_seen` property will be updated appropriately.

1.4.1 Resolution

Locations have full 6-digit decimal latitude and longitude components allowing high-precision placement on any map renderer or GIS software you wish to link them to. While the primary intention of locations is to provide a way of grouping a number of assets together into a single plant or facility, it is possible to create and manage a location for each individual asset if that is more suitable.

1.4.2 Data members

As with all Jitsuin Archivist data types, locations support custom attributes which can be defined and used for any purpose by the user. This enables storage of a mailing address, phone number, or contact details of the site manager, for example.

Natively, the data structure contains and requires only the basic identification and latitude/longitude information to plot and display in a GIS framework such as Google Maps.

For further details, see [Location Creation](#).

1.5 Access Policies

Sharing the right amount of information with your Value Chain partners is critical to creating a trustworthy Shared Service History for assets. It is important that every participant be able to see and contribute to the management of assets they have shared responsibility for without compromising security, commercial, or private personal information.

For example, competing vendors should not see each other's information, but both should be able to freely collaborate with their mutual customer or industry regulator.

In other scenarios, it is desirable to share basic maintenance information with a vendor or external maintenance company, whilst keeping critical operating information such as run cycles and cyber SLAs within a much smaller group.

Access Policies are the method through which this access is defined, allowing asset owners to collaborate with just the right partners at the right time, sharing as much or as little access to Shared Service Histories as the needs of the value chain partners dictate.

Note: To collaborate with a value chain partner you first need to exchange public keys in an out-of-band process such as email, and import the partner's keys into your system as an IAM Subject.

1.5.1 Configuring Access Policies

When To Create An Access Policy

All transactions are private by default, meaning that only the asset owner can see and update asset Service Histories until a sharing policy has been set up. This ensures ready compliance with important regimes such as GDPR and antitrust regulations, as well as allowing safe and ready collaboration with a large and diverse range of value chain partners in the Jitsuin network when required.

This means that Access Policies must be defined as soon as a need is identified in order to facilitate collaboration between value chain partners.

Considerations

As with any system handling large amounts of important data, it is important to carefully consider the design and scope of your access policy rules.

Every situation is different, and the Jitsuin Archivist Access Policy system is flexible and powerful enough to support most situations, but in general it is recommended to follow some basic rules:

- Aim for fewest possible number of policies: This makes it much easier to review and manage access rights.
- Balance complex, highly-specific policies with simple, broad ones: Rights granted by policy are additive.

- A single Access Policy can contain several Permission Groups, so it is possible to define a single filter to cover a particular population of assets, then apply different rights to different Subjects. This is often a simpler way to manage access than to create separate Access Policies for each set of Subjects.
- Remember attributes can change: ABAC policies are applied at time of access request, not at time of creation, so changing attributes on an asset may change which access policies apply to it. This is one of the primary advantages to an ABAC system but still needs to be borne in mind when designing access control processes.

Revoking Access

Jitsuin Archivist adopts a ‘default deny’ approach so access to an Asset Record is only possible if an Access Policy explicitly allows it.

Revoking access can therefore be achieved in a number of ways, any of which may be more or less appropriate for the circumstances:

- Remove the whole Access Policy
- Change the attributes of the Asset so that it no longer matches the Access Policy (eg change location)
- Change the attributes of the user or Subject so that they no longer match the Access Policy (eg change IDP group)
- Turn off the user’s login at the IDP

1.5.2 Policy Definition

Jitsuin Archivist employs a principle called [Attribute-Based Access Control](#).

Rather than applying a specific policy to each asset, or grouping them into rigid hierarchies, policies are defined in terms of the observable properties (or *attributes*) of assets and users, and if both match, the policy is applied. This enables much greater flexibility and expressivity than traditional hierarchical or role-based methods, whilst at the same time reducing complexity in handling large-scale systems.

Archivist Access Policies comprise 4 main parts:

- *Attribute Filters* and *Subjects*, which determine which users and assets the policy will apply to; and
- *Behaviours* and *Share Attributes*, which specify the read and write access granted to Archivist Asset Records under this policy

Attribute Filters

Attribute Filters are a set of attributes to test against asset attributes. This policy will apply to all assets that match. The filters take the form of a list of lists, where at least one attribute in each list must match the asset attributes. Or in other words, inner lists are OR, while outer lists are AND. For example:

```
Filters = [  
  [ListOne, ...],  
  [ListTwo, ...],  
  [ListThree, ...]  
]
```

In the above simplified example, the policy will apply to an asset only if the asset’s attributes match *at least one* entry from each of ListOne *AND* ListTwo *AND* ListThree.

In more detail:

```
Filters = [  
  [type=Pump, type=Valve],  
  [vendor=SynsationIndustries],  
  [location=ChicagoWest, location=ChicagoEast]  
]
```

In this case, the policy would apply to any Pump or Valve from SynsationIndustries installed in the ChicagoWest or ChicagoEast sites. It would not match:

- Other device types from SynsationIndustries;
- Pumps or Valves from any other vendor;
- SynsationIndustries Valves installed in a different location

Subjects

Subjects specify which users or value chain partners will be granted the policy access rights. These must have been imported as IAM Subjects before creating the policy.

Note: In future releases of Archivist it will be possible to specify qualifying Access Subjects by attribute. Presently each individual Subject must be included in the policy.

Behaviours

Behaviours specifies which asset behaviours the *Subjects* will be allowed to use. This can be considered equivalent to *write access*

Note: See [Behaviours](#) for details of behaviours.

An organization's ability to contribute to the Shared Service History for a given asset will be the union of all *Behaviours* write access granted under all policies.

Share Attributes

Share Attributes specify which attributes of the asset will be made visible to the specified *Subjects*. This can be considered equivalent to *read access*.

Note: See [Behaviours](#) for details of system-reserved `arc_*` attributes which should generally be shared with all Subjects.

An organization's copy of the Shared Service History for a given asset will contain the union of all *Share Attributes* read access granted under all policies.

1.6 Compliance Policies

In order to make these trust decisions, Jitsuin Archivist can be configured with *Compliance Policies* to check assets against. These policies specify things like tolerance for vulnerability windows, or SLAs for open maintenance calls. For example:

“Assets must be patched within 40 days of vulnerability notification”

“Maintenance calls must be answered within 72 hours”

Individual assets either pass or fail, and organizations can calculate their overall security/compliance posture based on what proportion of their assets are breaching their policy set.

Compliance signals can also be used to identify where risk lies in an organization and help to prioritize remedial activities.

1.6.1 User-defined Compliance Policies

Note: User-defined compliance policies are currently not supported. Future versions of Jitsuin Archivist will allow users to create policies that are best tuned to their business needs

1.6.2 Built-in Compliance Policies

Every Jitsuin Archivist implements a default policy in `compliance_policies/0000-0000-0000000000-00000000` which checks for outstanding firmware vulnerabilities or maintenance requests. If any requests are outstanding then the asset fails compliance. If all reported vulnerabilities have been patched and maintenance requests serviced then it passes.

For further details on the API for compliance posture, see [Compliance](#).

ADMINISTRATOR CONFIGURATION

The Jitsuin Archivist is fully integrated with the [Microsoft Identity Platform](#) (v2.0) using OAuth 2.0 and Open ID Connect. A small amount of Azure administrative configuration is required to control user assignment and enable non interactive API access. The following section details the necessary configuration.

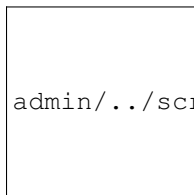
2.1 Register your Azure Active Directory with Jitsuin

To obtain access to your Jitsuin Archivist:

1. Communicate your Azure Directory ID to Jitsuin.
2. Receive a link to your Jitsuin Archivist.

2.1.1 Open Azure Active Directory

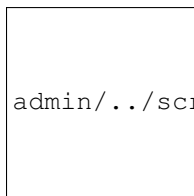
Browse to portal.azure.com and open the Azure Active Directory service from the portal menu.



admin/./screenshots/userdocs_aad_open_active_directory.png

2.1.2 Locating your Tenant ID

Locate the Tenant ID displayed on the Active Directory Overview blade and send this to Jitsuin.



admin/./screenshots/userdocs_aad_tenant_id.png

Note: This identifier is not sensitive information.

2.2 Assign Users for Interactive Use

To enable access for individual users to your Jitsui Archivist:

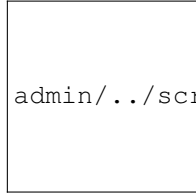
1. Assign Users to the Jitsui Archivist Enterprise application.
2. Grant assigned users the appropriate Jitsui Archivist roles.

This Microsoft guide provides the general details for [assigning users](#) and their roles.

The key steps are as follows:

2.2.1 Locate your Jitsui Archivist Enterprise Application

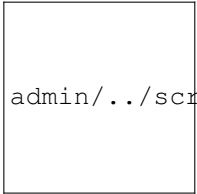
Having completed admin consent, locate the enterprise application principals for your Jitsui Archivist. There will be two **Homepage URLs** which match the FQDN for the link you received. The URL for the root resource is your API



admin/../../screenshots/userdocs_aad_ent

principal. This is where roles are assigned to users and non-interactive clients.

Note: The /webgate principal authorises your Jitsui Archivist deployment to act on your users behalf. It needs no further configuration.

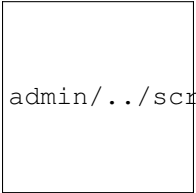


admin/../../screenshots/userdocs_aad_select_princ

Select the principal with the Homepage URL matching your link.

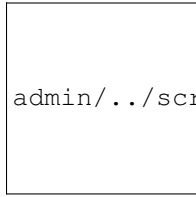
2.2.2 Check that User assignment is required for your Jitsui Archivist

The **User assignment required** setting must be **yes** in order to restrict access to particular users in your directory. If it is set to **no** any user at your organisation will be able to login.



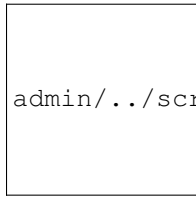
admin/../../screenshots/userdocs_aad_check_user_assignment.png

2.2.3 Add user to the enterprise application



admin/../../screenshots/userdocs_aad_add_user.png

2.2.4 Select the user for role assignment



admin/../../screenshots/userdocs_aad_select_user.png

2.3 Configure Client Credentials for Non-Interactive Access

To enable non-interactive access to Jitsuin Archivist APIs:

1. Create an Application registration in your Azure Active Directory.
2. Grant an API access permission for the registration referring to the Jitsuin Archivist API
3. Create a client secret

Note: Certificate based assertion of identity is fully supported. See **client_assertion_type** and **client_assertion** in the official [Azure documentation](#)

2.3.1 Create an Application registration

- Choose any name you like.
- Account type should be: **accounts in this organisational directory only**.
- Redirect URI - leave blank.

The top screenshot shows the Microsoft Azure portal interface for the 'jitsuin-system-test' tenant. The left sidebar contains navigation links for 'Create a resource', 'Home', 'Dashboard', 'All services', and 'FAVORITES'. The main content area displays the 'App registrations' overview page, which includes a search bar, a list of management options (Overview, Getting started, Security, Users, Groups, etc.), and a table of applications. The table currently shows 'No results'.

The bottom screenshot shows the 'Register an application' form. The 'Name' field is filled with 'jitsuin archivist api'. Under 'Supported account types', the option 'Accounts in this organizational directory only (jitsuin-system-test only - Single tenant)' is selected. The 'Redirect URI (optional)' field is set to 'Web' with a placeholder example 'e.g. https://myapp.com/auth'.

The Microsoft [quickstart register app](#) guide covers the general process.

2.3.2 Add an API Permission to the Application registration

Your app registration must be granted access to the Jitsuin Archivist API.

The screenshot displays the 'jitsuin-archivist-api - API permissions' page. On the left, a sidebar contains navigation links: Overview, Quickstart, Manage (Branding, Authentication, Certificates & secrets, API permissions, Expose an API, Owners, Roles and administrators (Previous), Manifest), and Support + Troubleshooting. The main content area is divided into three sections: 'API permissions', 'Grant consent', and 'Request API permissions'.

API permissions: This section explains that applications are authorized to call APIs when they are granted all the permissions the application needs. It includes an 'Add a permission' button and a table of permissions for 'Microsoft Graph (1)'. The table has columns for 'API / Permissions name' and 'Type'. The listed permission is 'User.Read' with a 'Delegate' type. Below the table, it states: 'These are the permissions that this application requests statable permissions dynamically through code. See best practice'.

Grant consent: This section states: 'These permissions have been granted for undefined but are permissions, you should consider adding them to the config'. It includes a button: 'Grant admin consent for jitsuin-system-test'.

Request API permissions: This section shows a list of APIs and a table of application permissions. The table has columns for 'Name' and 'Application (client) ID'. The listed APIs are: 'localhost.robin-desktop-dev-robin-avid-not-a-real.dns.domain.dev.eng.jitsuin.io/w...', 'Windows Azure Active Directory', 'Signup', 'IAM Supportability', 'Windows Azure Service Management API', 'dev-robin-1-avid.engineering-k8s-scratch-1.dev.eng.jitsuin.io/webgate', 'localhost.robin-desktop-dev-robin-avid-not-a-real.dns.domain.dev.eng.jitsuin.io', 'dev-robin-1-avid.engineering-k8s-scratch-1.dev.eng.jitsuin.io', 'Microsoft Azure App Service', and 'Policy Administration Service'.

The **Application permissions** enable access to the Jitsuin Archivist API using client secrets or certificates. The Microsoft [quickstart configure web app access](#) guide covers the general process. For non-interactive use see **Application permissions**.

2.3.3 Enable the desired Jitsuin Archivist roles

The screenshot displays the 'jitsuin-archivist-api - API permissions' page, focusing on the 'Request API permissions' section. The 'API permissions' section is visible on the left, showing the 'User.Read' permission. The 'Request API permissions' section shows a list of APIs and a table of application permissions. The 'Application permissions' section is highlighted with a dashed blue box, showing a table of permissions for 'dev-robin-1-avid.engineering-k8s-scratch-1.dev.eng.jitsuin.io'.

Request API permissions: This section shows a list of APIs and a table of application permissions. The table has columns for 'Name' and 'Application (client) ID'. The listed APIs are: 'localhost.robin-desktop-dev-robin-avid-not-a-real.dns.domain.dev.eng.jitsuin.io/w...', 'Windows Azure Active Directory', 'Signup', 'IAM Supportability', 'Windows Azure Service Management API', 'dev-robin-1-avid.engineering-k8s-scratch-1.dev.eng.jitsuin.io/webgate', 'localhost.robin-desktop-dev-robin-avid-not-a-real.dns.domain.dev.eng.jitsuin.io', 'dev-robin-1-avid.engineering-k8s-scratch-1.dev.eng.jitsuin.io', 'Microsoft Azure App Service', and 'Policy Administration Service'.

Application permissions: This section shows a table of permissions for 'dev-robin-1-avid.engineering-k8s-scratch-1.dev.eng.jitsuin.io'. The table has columns for 'Permission' and 'Admin Consent Required'. The listed permissions are: 'avid_admin_user', 'avid_admin_user', 'avid_default_user', and 'avid_default_user'. The 'Admin Consent Required' column shows 'Yes' for all listed permissions.

2.3.4 Grant administrator consent for the new Application registration

Home > jitsuin-system-test - App registrations > jitsuin archivist api - API permissions

jitsuin archivist api - API permissions

Do you want to grant consent for the requested permissions for all accounts in jitsuin-system-test? This will update any existing admin consent records this application already has to match what is listed below.

API / Permissions name	Type	Description	Admin Consent Required	Status
Microsoft Graph (1)				
User.Read	Delegated	Sign in and read user profile	-	
dev-robin-1-avid.engineering-k8s-scratch-				
avid_admin_user	Application	avid_admin_user	Yes	⚠ Not granted for jitsuin-sy...

These are the permissions that this application requests statically. You may also request user consent-able permissions dynamically through code. [See best practices for requesting permissions](#)

Grant consent

These permissions have been granted for undefined but aren't in the configured permissions list. If your application requires these permissions, you should consider adding them to the configured permissions list.

Home > jitsuin-system-test - App registrations > jitsuin archivist api - API permissions

jitsuin archivist api - API permissions

✓ Successfully granted admin consent for the requested permissions.

API permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs.

API / Permissions name	Type	Description	Admin Consent Required	Status
Microsoft Graph (1)				
User.Read	Delegated	Sign in and read user profile	-	✓ Granted for jitsuin-syste...
dev-robin-1-avid.engineering-k8s-scratch-				
avid_admin_user	Application	avid_admin_user	Yes	✓ Granted for jitsuin-syste...

These are the permissions that this application requests statically. You may also request user consent-able permissions dynamically through code. [See best practices for requesting permissions](#)

2.3.5 Add a client secret to the Application registration

jitsuin archivist api - Certificates & secrets

Credentials enable applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Certificates

Certificates can be used as secrets to prove the application's identity when requesting a token. Also can be referred to as public keys.

No certificates have been added for this application.

Thumbprint	Start Date	Expires
------------	------------	---------

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

Description	Expires	Value
-------------	---------	-------

No client secrets have been created for this application.

Take note of the client secret and the application object id (uuid).

Note: If you need to have different secrets for different Jitsuin Archivist roles create an application registration for each distinct set of roles.

2.4 Getting Access Tokens (client secret)

Having completed the steps at *Create an Application registration*, and taken note of its application id and the secret, a token can be obtained with the following command. Replace `${API_APP_ID}` with the application id, and `${API_APP_SECRET}` with your secret from the application registration. `${FQDN}` is the FQDN for your Jitsuin Archivist. `${TENANT}` is your directory id, see *Open Azure Active Directory*

```
$ RESPONSE=$(curl \
https://login.microsoftonline.com/${TENANT}/oauth2/token\
--data-urlencode "grant_type=client_credentials" \
--data-urlencode "client_id=${API_APP_ID}" \
--data-urlencode "client_secret=${API_APP_SECRET}" \
--data-urlencode "resource=https://${FQDN}")

$ TOKEN=$(echo -n $RESPONSE | jq .access_token | tr -d '"')
```

2.5 Testing Access

To confirm access token configuration, use the shell command (above) to obtain an access token. The response is json structured data. The token is found in the `access_token` field. It is a base64 encoded [JSON Web Token](#).

The header and payload of the TOKEN can be examined with the following commands

```
# Header
echo -n $TOKEN | cut -d '.' -f 1 | base64 -D

# Payload
echo -n $TOKEN | cut -d '.' -f 2 | base64 -D
```

Note: Decoding tokens with an online service exposes your Archivist until you delete the test secret.

The following python script demonstrates how to safely obtain and verify a token. The example requires python 3.6. Run the script like this:

```
python3 check-token.py -t ${TENANT} -c ${API_APP_ID} -s ${API_APP_SECRET} -f ${FQDN}
```

Copy the following python code to check-token.py

```
#!/usr/bin/env python3

# REQUIRES Python 3.6
import sys
import argparse
import subprocess as sp
```

(continues on next page)

(continued from previous page)

```

import urllib.parse
import base64
import json
import calendar
import datetime

verify_token=True
try:
    import jwcrypto
    import jwcrypto.jwk
    import jwcrypto.jwt
except ImportError:
    verify_token=False

def run():
    p = argparse.ArgumentParser( description=__doc__)

    p.add_argument("-T", "--token")
    p.add_argument("-t", "--tenant")
    p.add_argument("-c", "--client-id")
    p.add_argument("-s", "--client-secret")
    p.add_argument("-f", "--fqdn")

    args = p.parse_args()

    # Support checking a token provided 'as is' and also fetching and checking
    # a token using the expected customer configuration items

    token = args.token
    if token is None:
        secret = urllib.parse.quote(args.client_secret)
        resource = urllib.parse.quote("https://" + args.fqdn)

        data = f"grant_type=client_credentials&client_id={args.client_id}"
        data += f"&client_secret={secret}&resource={resource}"

        cmd = [
            "curl", "-X", "POST",
            "-HContent-Type: application/x-www-form-urlencoded",
            f"https://login.microsoftonline.com/{args.tenant}/oauth2/token",
            "-d", data]

        # Avoid the unpleasant curl output
        cp = sp.run(cmd, stdout=sp.PIPE, stderr=sp.PIPE, check=True)
        token = cp.stdout.decode()
        jdoc = json.loads(token)
        token = jdoc["access_token"]
        print("TOKEN:")
        print(token)

    header, payload, *sig = token.split('.')

    header = json.loads(base64.b64decode(header + "===").decode())
    print(json.dumps(header))

    payload = json.loads(base64.b64decode(payload + "===").decode())

```

(continues on next page)

(continued from previous page)

```

print(json.dumps(payload, indent=4, sort_keys=True))

# Check that the 'aud' field matches the resource
if args.fqdn and 'https://' + args.fqdn != payload["aud"]:
    print("Missing or unexpected aud", file=sys.stderr)
    return -1

# Check that its issued by the expected tenancy
if args.tenant and args.tenant not in payload["iss"]:
    print("Unexpected directory id in issuer (iss)", file=sys.stderr)

# Check the Jitsuin Archivist roles are present
roles = payload["roles"]
if "archivist_administrator" not in roles and "guest" not in roles:
    print("Token is missing the required roles", file=sys.stderr)
    return -1

# Check the freshly issued token has not expired and that the issue time is
# sensible
iat = int(payload["iat"])
exp = int(payload["exp"])
now = calendar.timegm(datetime.datetime.utcnow().utctimetuple())

if now < iat:
    print(f"iat before 'now'. iat={iat}, now={now}", file=sys.stderr)
    return -1
if now >= exp:
    print(
        f"now after 'exp', token expired "
        f"or invalid. now={now}, exp={exp}", file=sys.stderr)
    return -1

# Get the IdP Open ID configuration
cmd = [
    "curl", "-HAccept: application/json",
    f"{payload['iss']}/.well-known/openid-configuration"]
cp = sp.run(cmd, stdout=sp.PIPE, stderr=sp.PIPE, check=True)

oidconf = json.loads(cp.stdout.decode())

# Fetch the keys for verification
cmd = ["curl", "-HAccept: application/json", f"{oidconf['jwks_uri']}"]
cp = sp.run(cmd, stdout=sp.PIPE, stderr=sp.PIPE, check=True)

jwks = json.loads(cp.stdout.decode())
key = None
for k in jwks["keys"]:
    if k["kid"] == header["kid"]:
        key = k
        break
if key is None:
    print(
        "Failed to find token verification key at issuer", file=sys.stderr)
    return -1

if verify_token is False:
    print("Please install jwcrypto to verify your token")

```

(continues on next page)

(continued from previous page)

```
        return 0

    jwk = jwcrypto.jwk.JWK(**key)
    jwt = jwcrypto.jwt.JWT()
    # If there is any problem with the token, this function will raise an
    # exception.
    jwt.deserialize(token, key=jwk)

    return 0

if __name__ == "__main__":
    try:
        sys.exit(run())
    except json.decoder.JSONDecodeError as e:
        print(f"json decoding error {str(e)}")
    except sp.CalledProcessError as cpe:
        print(cpe.output, file=sys.stderr)
    except KeyError as e:
        print(f"expected key missing {str(e)}", file=sys.stderr)
    except ValueError as e:
        print(str(e), file=sys.stderr)
    except Exception as e:
        print(str(e), file=sys.stderr)
    sys.exit(-1)
```

Delete the test secret once this test is completed.

Note: Certificate based assertion of identity is fully supported. See `client_assertion_type` and `client_assertion` in the official [Azure documentation](#)

2.6 Using PingOne for Enterprise as IDP

This guide provides instructions on how to integrate PingOne for Enterprise IDP with JitsuIn Archivist.

Before starting:

- JitsuIn will provide a unique URL that will be used when creating the application in PingOne for Enterprise
- Note that some values from step 6 below will be needed by JitsuIn to complete the integration.
- Ping Identity also provide instructions for adding application in the following link: <https://docs.pingidentity.com/bundle/pingone/page/rkz1564020496702-1.html>


Steps:

1. Log into PingOne for Enterprise and navigate to “My Applications”
2. Add a new OIDC application and follow the steps below
3. Select application type “Web App” and click Next

Add OIDC Application

SELECT AN APPLICATION TYPE : Web App


Select the OIDC application type that best suites the type of application you want to integrate with PingOne.



Web applications that are accessed within a browser

- .NET web apps
- Java apps


WEB APP



Applications that are stored and run from a device or desktop

- IOS and Android apps
- Desktop apps


NATIVE APP



A front-end application that uses an API

- Angular
- Node JS

SINGLE PAGE APP



Apps configured by advanced users from the ground up

- Your Choice
- No barriers
- Complete Flexibility

ADVANCED CONFIGURATION

Cancel

Next

- (Wizard Step 1) Provide application details as desired (i.e. Application name: Archivist)

Add OIDC Application

SELECT AN APPLICATION TYPE : Single Page App Edit

1 PROVIDE DETAILS ABOUT YOUR APPLICATION

This information will be displayed on the PingOne dock for your end users.

APPLICATION NAME ?

SHORT DESCRIPTION

Describe your application in 1000 characters or less

CATEGORY ?

Choose One ▼

ADD APPLICATION GRAPHICS

ICON ?

Maximum size is 1MB

5. (Wizard Step 2) Configure authorization settings

1. Enable “Refresh Token”
2. Click the “Add Secret” button
3. Send the client ID and client secret to Jitsuin. These are needed to complete the integration in the hosted Archivist application
4. Leave other values default or set as desired

Add OIDC Application

2

AUTHORIZATION SETTINGS

ACCESS TOKEN

OVERWRITE ACCESS TOKEN LIFETIME ?
☐

GRANTS

ALLOWED GRANT TYPES ?
☒ Authorization Code
☐ Refresh Token

CREDENTIALS

CLIENT ID ?
825f1513-1a40-4e01-a49e-de48f054b19b

+ Add Secret

INTEGRATION DETAILS

6. (Wizard Step 3) Enter the unique URLs as provided by Jitsuin

Add OIDC Application

3 SSO FLOW AND AUTHENTICATION SETTINGS

START SSO URL ?

REDIRECT URIS ?

+ Add URL

LOGOUT URL ?

AUTHENTICATION

☐ Force authentication ?

☐ Force multi-factor authentication ?

Cancel

Next

4 DEFAULT USER PROFILE ATTRIBUTE CONTRACT

5 CONNECT CODES

7. (Wizard Step 4) Click Next

Add OIDC Application

4

DEFAULT USER PROFILE ATTRIBUTE CONTRACT

Configure additional attributes returned by the UserInfo endpoint when the initial authentication request includes the openid scope.

ATTRIBUTE NAME ?

sub

REQUIRED ?

☒

idpid

REQUIRED ?

☒

+ Add Attribute

Cancel

Next

5

CONNECT SCOPES

6

ATTRIBUTE MAPPING

7

GROUP ACCESS

8. (Wizard Step 5) Add Email scope to Connected scopes

Add OIDC Application

5

CONNECT SCOPES

Select the scopes that are allowed to be requested for this application during authentication.

LIST OF SCOPES

OpenID Profile (profile)

OpenID Email (email)

OpenID Address (address)

OpenID Phone (phone)

CONNECTED SCOPES 1

OpenID Connect (openid)

Cancel

Next

6

ATTRIBUTE MAPPING

7

GROUP ACCESS

9. (Wizard Step 6) Select “Email” for email attribute mapping and “Id” for sub attribute mapping

Add OIDC Application

CONNECT SCOPES

Edit

6

ATTRIBUTE MAPPING

Map your identity repository attributes to the claims available to this application. By default, the attribute mapping will inherit the attribute mapping specified in the OAuth settings. Modify the attribute mapping specifically for this application as necessary. The claims listed here include all claims from the access token attribute contract, the UserInfo attribute contract for this application and any scopes allowed by this application.

email

Email

Advanced

email_verified

Advanced

sub

Id

Advanced

Cancel

Next

7

GROUP ACCESS

10. (Wizard Step 7) Add appropriate groups to allow desired access for users

Add OIDC Application

CONNECT SCOPES

ATTRIBUTE MAPPING

7

GROUP ACCESS

Select all user groups that should have access to this application. Users that are members of the added groups will be able to SSO to this application and will see this application on their personal dock.

Search

AVAILABLE GROUPS

dev@directory

+

Domain Administrators@directory

+

ADDED GROUPS 1

Users@directory

-

Cancel

Done

Once Jitsui has received the client ID and secret the connection will be completed and the Archivist app will be available on the URL provided using SSO login credentials.

API REQUEST AUTHORIZATION AND AUTHENTICATION

Authorization and Authentication of individual Jitsui Archivist API requests uses **Bearer tokens**

See *Getting Access Tokens (client secret)* for details on how to obtain the token. And *Configure Client Credentials for Non-Interactive Access* for the necessary administrative configuration.

The bearer token should be stored in a file and an environment variable **BEARER_TOKEN_FILE** contains the name of the file.

The text in the BEARER_TOKEN_FILE should follow the format:

[illegible]

where the x's are replaced by the actual contents of the bearer token.

Note: Recommended that the directory containing the **BEARER_TOKEN_FILE** have 0600 permissions

Note: Certificate based assertion of identity is fully supported. See “client_assertion_type” and “client_assertion” in the official [Azure documentation](#)

TENANCIES API

4.1 Tenancy Information

The tenancies service provides information about your Archivist tenancy.

The tenancy information includes the list of user principals who have *root* or super-user access rights.

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Note: Only tenant *root* users are allowed to call the tenancies endpoint. Other users will receive a 403 response.

4.1.1 Fetch the current list of tenant root principals

To fetch the list of root principals, simply GET the `tenancies/root_principals` resource:

```
$ curl -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v1/tenancies/root_principals
```

4.1.2 Update the list of tenant root principals

Define the update parameters and store in `/path/to/jsonfile`:

```
{
  "root_principals": [
    {
      "issuer": "https://login.microsoftonline.com/5c129635-5858-4fe3-9bef-
↪444f6c7ee1cf/v2.0",
      "subject": "58589bef-4fe3-9a3b-23df-8527bc45e1cf",
```

(continues on next page)

(continued from previous page)

```
        "display_name": "Jane Smith",
        "email": "jane.smith@synsation.org"
    },
    {
        "issuer": "https://login.microsoftonline.com/5c129635-5858-4fe3-9bef-
↪444f6c7eelcf/v2.0",
        "subject": "27bc5b4f-9a3b-4fe3-23df-e1c7bc45e1cf",
        "display_name": "Nate Rogers",
        "email": "nate.rogers@synsation.org"
    }
}
```

Note:

issuer required The principal's issuer string for your Identity Provider. This must match the Identity Provider for all existing root principals.

subject required The principal's subject string as provided by your Identity Provider.

display_name optional Friendly name for the user principal. Displayed in the Archivist GUI.

email optional Email address for the principal.

Update the root principals by PATCHing the `tenancies/root_principals` resource:

```
$ curl -v -X PATCH \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/v1/tenancies/root_principals
```

Note: For safety reasons you are not allowed to remove yourself from the list of root principals.

TLS CA CERTIFICATES MANAGEMENT (V1)

Organisational access to the various resources in Jitsuin Archivist can be managed by defined TLS CA certificates. The contents of the PEM file is stored together with a display name and a unique id.

5.1 TLS CA Certificates Upload (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://symsation.1234-5678.nodes.archivist.jitsuin.io
```

Define the TLS CA certificate parameters and store in /path/to/jsonfile (certificate field shortened for brevity):

```
{
  "display_name": "Some description",
  "certificate": "-----BEGIN CERTIFICATE-----\n
↪nMIIFxDCCA6ygAwIBAgIBAgJANBgkqhkiG9w0BAQsFADCBsDELMakGA1UEBhMCVVMx\NETAPBgNV...1NF/\n
↪BjDZ4wdexw==\n-----END CERTIFICATE-----\n"
}
```

To include the PEM file content in a JSON string it must be flattened to a single line. To create a single line representation of a PEM file for the archivist api, you must replace new lines with the literal string “n”. The following unix command could be used:

```
$ awk 'NF {sub(/\r/, ""); printf "%s\\n",$0;}' cert-name.pem
```

Note:

display_name required Friendly name for the location. Displayed in the Archivist GUI.

certificate required Single line “flattened” PEM containing a CERTIFICATE.

Create the CA Certificate:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/v1/tlscacertificates
```

The response is (certificate field shortened for brevity):

```
{
  "identity": "tlscacertificates/3f5be24f-fd1b-40e2-af35-ec7c14c74d53",
  "display_name": "Some description",
  "certificate": "-----BEGIN CERTIFICATE-----
→MIIEBDCCAuygAwIBAgIDAjppMA0GCSqGSIB3DQEBBQUAMEIxCzAJBgqNVBAYTA1VT -----END
→CERTIFICATE-----"
}
```

Note: A full API reference is available in [Swagger POST API](#)

5.2 TLS CA Certificate Retrieval (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

TLS CA Certificate records in Jitsuin Archivist are tokenized at creation time and referred to in all API calls and smart contracts throughout the system by a unique identity of the form:

```
tlscacertificates/12345678-90ab-cdef-1234-567890abcdef.
```

If you do not know the certificate's identity you can fetch TLS CA Certificate records using other information you do know, such as the certificate's name.

5.2.1 Fetch all TLS CA Certificates (v1)

To fetch all TLS CA certificates records, simply GET the `tlscacertificates` resource:

```
$ curl -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v1/tlscacertificates
```

5.2.2 Fetch specific TLS CA Certificate by identity (v1)

If you know the unique identity of the TLS CA certificate Record simply GET the resource:

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v1/tlscacertificates/6a951b62-0a26-4c22-a886-1082297b063b
```

5.2.3 Fetch TLS CA Certificates by name (v1)

To fetch all TLS CA Certificates with a specific name, GET the `tlscacertificates` resource and filter on `display_name`:

```
$ curl -g -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v1/tlscacertificates?display_name=Acme
```

Each of these calls returns a list of matching TLS CA Certificate records in the form (certificate field shortened for brevity):

```
{
  "certificates": [
    {
      "identity": "tlscacertificates/6a951b62-0a26-4c22-a886-1082297b063b",
      "display_name": "Some description",
      "certificate": "-----BEGIN CERTIFICATE-----
→MIIEBDCCAuygAwIBAgIDAjppMA0GCSqGSIb3DQEBBQUAMEIxCzAJBgNVBAYTA1VT -----END
→CERTIFICATE----- "
    },
    {
      "identity": "tlscacertificates/12345678-0a26-4c22-a886-1082297b063b",
      "display_name": "Some other description",
      "certificate": "-----BEGIN CERTIFICATE-----
→XYZEBDCCAuygAwIBAgIDAjppMA0GCSqGSIb3DQEBBQUAMEIxCzAJBgNVBAYTA1VT -----END
→CERTIFICATE----- "
    }
  ]
}
```

Note: The number of records returned has a maximum limit. If this limit is too small then one must use *API Request Paging*.

Note: The total number of certificates that exist is returned in the response header field 'x-total-count' if the 'x-request-total-count' header on the request is set to 'true'. The curl option '-i' will emit this to stdout.

Note: A full API reference is available in [Swagger GET API](#)

5.3 TLS CA Certificate Deletion (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://symsation.1234-5678.nodes.archivist.jitsuin.io
```

To delete a TLS CA Certificate, issue following request:

```
$ curl -v -X DELETE \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  $URL/archivist/v1/tlscacertificates/47b58286-ff0f-11e9-8f0b-362b9e155667
```

The response is:

```
{ }
```

Note: A full API reference is available in [Swagger POST API](#)

5.4 TLS CA Certificates Update (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://symsation.1234-5678.nodes.archivist.jitsuin.io
```

Define the TLS CA certificates parameters to be changed and store in /path/to/jsonfile:

```
{
  "display_name": "new description"
}
```

Note:

display_name descriptive name of TLS CA certificate

Update the TLS CA Certificate:


```
$ curl -v -X PATCH \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/v1/tlscacertificates/47b58286-ff0f-11e9-8f0b-362b9e155667
```

The response is (certificate field shortened for brevity):

```
{
  "identity": "tlscacertificates/3f5be24f-fd1b-40e2-af35-ec7c14c74d53",
  "display_name": "Some description",
  "certificate": "-----BEGIN CERTIFICATE-----
↳MIIEBDCCAuygAwIBAgIDAjppMA0GCSqGSIb3DQEBBQUAMEIxCzAJBgqNVBAYTA1VT -----END
↳CERTIFICATE-----"
}
```

Note: A full API reference is available in [Swagger POST API](#)

5.5 TLS CA Certificates Swagger API

GET /archivist/v1/tlscacertificates

List TLS CA certificates

Returns a paginated list of TLS CA certificates

Query Parameters

- **order_by** (*string*) – Specify the sort order for the results.
- **page_size** (*integer*) – Maximum entries per page
- **page_token** (*string*) – The next_page_token returned from a previous list request if any.
- **display_name** (*string*) – Further fields are bound to query parameters and act to filter the result

Customer friendly name for the TLS CA certificate.

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to read the TLS CA certificate.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **next_page_token** (*string*) – Token to retrieve the next page of results or empty if there are none.
- **tlscacertificates[] .display_name** (*string*) – Customer friendly name for the TLS CA certificate.
- **tlscacertificates[] .identity** (*string*) – Unique identification for the TLS CA certificate, Relative Resource Name
- **tlscacertificates[] .tlscacertificate** (*string*) – TLS CA Certificate as base64 string representing the contents of the PEM file

POST /archivist/v1/tlscacertificates
Upload a TLS CA certificate

This request uploads a TLS CA certificate. The display_name is the friendly name.

Request JSON Object

- **display_name** (*string*) – Customer friendly name for the TLS CA certificate. (required)
- **tlscacertificate** (*string*) – TLS CA Certificate as string representing the contents of the PEM file (required)

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **402 Payment Required** – Returned when the user's quota of certificates has been reached.
- **403 Forbidden** – Returned when the user is not authorized to create a tlscacertificate.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **display_name** (*string*) – Customer friendly name for the TLS CA certificate.
- **identity** (*string*) – Unique identification for the TLS CA certificate, Relative Resource Name
- **tlscacertificate** (*string*) – TLS CA Certificate as base64 string representing the contents of the PEM file

GET /archivist/v1/tlscacertificates/{uuid}
Get a TLS CA certificate

Returns the identified tlscacertificate

Parameters

- **uuid** (*string*) – Specify the TLS CA Certificate UUID where *tlscacertificates/{uuid}* is the TLS CA Certificate Identity e.g. *08838336-c357-460d-902a-3aba9528dd22* from Identity *tlscacertificates/08838336-c357-460d-902a-3aba9528dd22*

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to read the TLS CA certificate.
- **404 Not Found** – Returned when the identified TLS CA certificate does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **display_name** (*string*) – Customer friendly name for the TLS CA certificate.
- **identity** (*string*) – Unique identification for the TLS CA certificate, Relative Resource Name
- **tlscacertificate** (*string*) – TLS CA Certificate as base64 string representing the contents of the PEM file

DELETE /archivist/v1/tlscacertificates/{uuid}

Delete a TLS CA Certificate

Delete the identified TLS CA Certificate

Parameters

- **uuid** (*string*) – Specify the TLS CA Certificate UUID where *tlscacertificates/{uuid}* is the TLS CA Certificate Identity e.g. *08838336-c357-460d-902a-3aba9528dd22* from Identity *tlscacertificates/08838336-c357-460d-902a-3aba9528dd22*

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to delete the TLS CA certificate.
- **404 Not Found** – Returned when the identified laccess policy does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

PATCH /archivist/v1/tlscacertificates/{uuid}

Update a TLS CA certificate's details

Perform a full or partial update of the identified TLS CA certificate

Parameters

- **uuid** (*string*) – Specify the TLS CA Certificate UUID where *tlscacertificates/{uuid}* is the TLS CA Certificate Identity e.g. *08838336-c357-460d-902a-3aba9528dd22* from Identity *tlscacertificates/08838336-c357-460d-902a-3aba9528dd22*

Query Parameters

- **mask** (*string*) – The mask says exactly which fields are to be updated. This removes the ambiguity caused by ‘zero’ valued fields. <https://grpc-ecosystem.github.io/grpc-gateway/docs/patch.html>

Request JSON Object

- **display_name** (*string*) – Customer friendly name for the TLS CA certificate.
- **identity** (*string*) – Unique identification for the TLS CA certificate, Relative Resource Name
- **tlscacertificate** (*string*) – TLS CA Certificate as base64 string representing the contents of the PEM file

Status Codes

- 200 OK – A successful response.
- 400 Bad Request – Returned when the request is badly formed.
- 401 Unauthorized – Returned when the user is not authenticated to the system.
- 403 Forbidden – Returned when the user is not authorized to update the TLS CA certificate.
- 404 Not Found – Returned when the identified TLS CA certificate does not exist.
- 429 Too Many Requests – Returned when a user exceeds their subscription’s rate limit for requests.
- 500 Internal Server Error – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **display_name** (*string*) – Customer friendly name for the TLS CA certificate.
- **identity** (*string*) – Unique identification for the TLS CA certificate, Relative Resource Name
- **tlscacertificate** (*string*) – TLS CA Certificate as base64 string representing the contents of the PEM file

GET /archivist/v1/tlscacertificates:caps
Get remaining capped resources for TLSCACertificates

Not stable or officially supported. Get remaining capped resources for TLSCACertificates

Status Codes

- 200 OK – A successful response.
- 401 Unauthorized – Returned when the user is not authenticated to the system.
- 429 Too Many Requests – Returned when a user exceeds their subscription’s rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **caps[] .resource_remaining** (*string*) – Number of capped resources remaining
- **caps[] .resource_type** (*string*) – String identifying the capped resource type

GET /archivist/v1/tlscacertificates:openapi
Get OpenAPI spec for TLSCACertificates

Get OpenAPI v2.0 spec for TLSCACertificates

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

GET /archivist/v1/tlscacertificates:openapi-ui
Get OpenAPI UI for TLSCACertificates

Get OpenAPI v2.0 UI for TLSCACertificates

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

COMPLIANCE API

The compliance API provides a “trustworthiness” signal for an Asset based on compliance, or otherwise, with Compliance Policies.

6.1 Compliance

By maintaining a complete traceable record of When Who Did What to a Thing, RKVST makes it possible for any authorized stakeholder to quickly and easily verify that critical processes have been followed and recorded correctly. And if they weren’t, the record makes it easy to discover where things went wrong and what to fix.

For instance: missed or late maintenance rounds can be detected simply by spotting gaps in the maintenance record; cyber vulnerable devices can be found by comparing ideal baselines with patching records; out-of-order process execution and handling violations are visible to all; and back-dating is automatically detectable.

All of this is very valuable in audit and RCA situations after an incident, where there is time to collect together Asset Records, piece together the important parts, and analyse the meaning.

But what if the same information could be used for real-time decision-making that might avert an incident?

This is where RKVST’s “compliance posture” APIs come in. These take the thinking and processing burden off the client by providing a single, simple API call to answer the complex question: “given all you know about this asset, should I trust it right now?”. Additionally, and crucially for sensitive use cases, the yes or no answer comes with a detailed defensible reason why, which can be inspected by relevant stakeholders during or after the event.

When put all together, this enables high quality decision making based on the best available data, even giving confidence to automated or AI systems to play a full part in operations. Assets can be checked as part of access control logic, prior to accepting data or commands from them, prior to accepting a shipment, or anything else that is important to your business.

6.2 Creating Compliance Policies

Compliance Posture is measured against user-defined rule sets called Compliance Policies. Compliance policies are created once and then Assets can be tested against them at any point in time. For instance, a policy might state that “MaintenanceAlarm Events must be dealt with and a MaintenanceReport Event recorded with 72 hours”. This creates a Compliance Policy object in the system against which any asset can be tested as needed.

6.3 Types of Compliance Policies

RKVST allows users to define Compliance Policies of the following types:

6.3.1 COMPLIANCE_SINCE

This Compliance Policy checks if the time since the last occurrence of a specific Event Type has elapsed a specified threshold. For example “Time since last Maintenance must be less than 72 hours”:

```
{
  "compliance_type": "COMPLIANCE_SINCE",
  "description": "Maintenance should be performed every 72h",
  "display_name": "Regular Maintenance",
  "asset_filter": [
    { "or": ["attributes.arc_location_identity:locations/5eef2b71-35c1-
↪4376-a166-6c64bfa72f4b"]}
  ]
  "event_display_type": "Maintenance Performed",
  "time_period_seconds": "259200"
}
```

Note:

event_display_type Type of event we want to check with this compliance policy

time_period_seconds The maximum amount of time allowed since a specified event type last occurred in seconds

6.3.2 COMPLIANCE_CURRENT_OUTSTANDING

This Compliance Policy will only pass if there is an associated answering event addressing a specified outstanding event, for example defining pairs of Events like “Maintenance Request” and “Maintenance Performed”. To correlate events define the attribute “arc_correlation_id” in the Event Attributes and set it to the same value on each pair of events that are to be associated. For example, checking there are no outstanding “Maintenance Request” Events that are not addressed by an associated “Maintenance Performed” Event:

```
{
  "compliance_type": "COMPLIANCE_CURRENT_OUTSTANDING",
  "description": "There should be no outstanding Maintenance Requests",
  "display_name": "Outstanding Maintenance Requests",
  "asset_filter": [
    { "or": ["attributes.arc_location_identity:locations/5eef2b71-35c1-
↪4376-a166-6c64bfa72f4b"]}
  ]
  "event_display_type": "Maintenance Requests",
  "closing_event_display_type": "Maintenance Performed"
}
```

Note:

event_display_type Type of event that should be addressed by the event defined in closing_event_display_type

closing_event_display_type Type of event addressing the event defined in event_display_type

6.3.3 COMPLIANCE_PERIOD_OUTSTANDING

This Compliance Policy will only pass if the time between a pair of correlated events did not exceed the defined threshold. To correlate events define the attribute “arc_correlation_id” in the Event Attributes and set it to the same value on each pair of events that are to be associated. For example, a policy checking that the time between “Maintenance Request” and “Maintenance Performed” Events does not exceed the maximum 72 hours:

```
{
  "compliance_type": "COMPLIANCE_PERIOD_OUTSTANDING",
  "description": "There should be no outstanding Maintenance Requests",
  "display_name": "Outstanding Maintenance Requests",
  "asset_filter": [
    { "or": ["attributes.arc_location_identity:locations/5eef2b71-35c1-4376-a166-6c64bfa72f4b"]}
  ]
  "event_display_type": "Maintenance Requests",
  "closing_event_display_type": "Maintenance Performed",
  "time_period_seconds": "259200"
}
```

Note:

event_display_type Type of event that should be addressed by the event defined in closing_event_display_type

closing_event_display_type Type of event addressing the event defined in event_display_type

time_period_seconds Maximum amount of time that can elapse between associated events in seconds

6.3.4 COMPLIANCE_DYNAMIC_TOLERANCE

This Compliance Policy will only pass if the time between a pair of correlated events did not exceed the defined variability. To correlate events define the attribute “arc_correlation_id” in the Event Attributes and set it to the same value on each pair of events that are to be associated. For example, a policy checking that the time between “Maintenance Request” and “Maintenance Performed” Events in the last week does not exceed a variability of 0.5 standard deviations around the mean:

```
{
  "compliance_type": "COMPLIANCE_DYNAMIC_TOLERANCE",
  "description": "Average time between Maintenance Requested/Performed",
  "display_name": "outlying Maintenance Requests",
  "asset_filter": [
    { "or": ["attributes.arc_location_identity:locations/5eef2b71-35c1-4376-a166-6c64bfa72f4b"]}
  ]
  "event_display_type": "Maintenance Requests",
  "closing_event_display_type": "Maintenance Performed",
}
```

(continues on next page)

(continued from previous page)

```
}
  "dynamic_window": 604800,
  "dynamic_variability": 0.5
}
```

Note:

event_display_type Type of event that should be addressed by the event defined in closing_event_display_type

closing_event_display_type Type of event addressing the event defined in event_display_type

dynamic_window number of seconds in the past - only events in this time window are considered.

dynamic variability exceeding this number of standard deviations from the mean will cause compliance to fail for a particular pair of matching events..

6.3.5 COMPLIANCE_RICHNESS

This Compliance Policy will only pass if the assertions conducted on an attribute value pass. An assertion is comprised of: an attribute name, a comparison value and an operator to compare with; for example “rad<7”. The operator can be one of six relational operators: equal to, not equal to, greater than, less than, greater than or equal to, less than or equal to. [=|!=|>|<|>=|<=]. Assertions are comprised of two lists, an inner list and outer list. The inner list states that, if any of the assertions pass, then the list is compliant (OR logic). For example: {“or”: [“rad<7”, “rad=10”]}. The outer list states that, all inner lists need to be compliant in order for the policy to be compliant (AND logic).

Compliance is a signal, not a perfect answer. Therefore equivalence of floats is exact, not approximate.

```
{
  "compliance_type": "COMPLIANCE_RICHNESS",
  "description": "Rad level is less than 7"
  "display_name": "Rad limit",
  "asset_filter": [
    { "or": ["attributes.arc_location_identity:locations/5eef2b71-35c1-
↪4376-a166-6c64bfa72f4b"]}
  ],
  "richness_assertions": [
    { "or": ["rad<7"]}
  ],
}
```

Note:

richness_assertions The assertions to be made, against asset attributes, to check if the asset is compliant.

6.4 Compliance Policy Creation

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Create a Policy with:

```
$ curl -v -X POST \
  -H "$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "/path/to/jsonfile" \
  $URL/archivist/v1/compliance_policies
```

Using data from /path/to/jsonfile in the format described in #Types of Compliance Policies.

Sample response:

```
{
  "identity": "compliance_policies/6a951b62-0a26-4c22-a886-1082297b063b",
  "compliance_type": "COMPLIANCE_CURRENT_OUTSTANDING",
  "description": "There should be no outstanding Maintenance Requests",
  "display_name": "Outstanding Maintenance Requests",
  "asset_filter": [
    { "or": ["attributes.arc_location_identity:locations/5eef2b71-35c1-
↪4376-a166-6c64bfa72f4b"]}
  ]
  "event_display_type": "Maintenance Requests",
  "closing_event_display_type": "Maintenance Performed",
  "time_period_seconds": "259200"
}
```

6.5 Compliance Checking

The **compliancev1** endpoint reports on the status of an Asset's Compliance with Compliance Policies.

Query the endpoint:

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v1/compliance/assets/6a951b62-0a26-4c22-a886-1082297b063b
```

or if determining compliance at some historical date:

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v1/compliance/assets/6a951b62-0a26-4c22-a886-1082297b063b?
↪compliant_at=2019-11-27T14:44:19Z
```

The response is:

```
{
  "compliant": true,
  "compliance": [
    {
      "compliance_policy_identity": "compliance_policies/0000-0000-000000000-
↪00000000",
      "compliant": true,
      "reason": ""
    }
  ],
  "compliant_at": "2019-11-27T14:44:19Z"
}
```

Note:

compliant Overall compliance status, false if any Compliance Policy is not “compliant”

compliant_at Timestamp at which compliance was determined

The response contains a list of Compliance Statements.

Each member of the list has the following attributes:

compliance_policy_identity The identity of the Compliance Policy this statement refers to

compliant Compliance status for this Compliance Policy

reason description of non-compliance (only if compliant is false)

See [Swagger GET API](#)

6.6 Compliance Swagger API

GET `/archivist/v1/compliance/assets/{uuid}`

List all compliance status relevant to an asset

Parameters

- **uuid** (*string*) – Specify the Asset UUID where *assets/{uuid}* is the Asset Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *assets/add30235-1424-4fda-840a-d5ef82c4c96f*

Query Parameters

- **page_size** (*integer*) – Maximum results per page.
- **page_token** (*string*) – The next_page_token returned from a previous list request if any.
- **order_by** (*string*) – Specify the sort order for the results.
- **compliant_at** (*string*) – timestamp at which compliance is determined
time at which compliance is determined

Status Codes

- **200 OK** – A successful response.

- **206 Partial Content** – The number of compliance statements exceeds the servers limit.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to access the requested resource.
- **404 Not Found** – Returned when the asset with the id does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **compliance[].compliance_policy_identity** (*string*) – identity of the compliance policy (read only)
- **compliance[].compliant** (*boolean*) – status of compliance against the compliance policy
- **compliance[].reason** (*string*) – reason for non-compliance (read only)
- **compliant** (*boolean*) – overall compliance status for the asset
- **compliant_at** (*string*) – time at which compliance is determined (read only)
- **next_page_token** (*string*) – Token to retrieve the next page of results or empty if there are none.

GET /archivist/v1/compliance_policies

Query Parameters

- **page_size** (*integer*) – Maximum results per page.
- **page_token** (*string*) – The next_page_token returned from a previous list request if any.
- **order_by** (*string*) – Specify the sort order for the results.
- **compliance_type** (*string*) – policy type
- **description** (*string*) – Customer description of the compliance policy.
- **display_name** (*string*) – Customer friendly name for the compliance policy.

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to access the requested resource.
- **404 Not Found** – Returned when the asset with the id does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **compliance_policies[].asset_filter[].or[]** (*string*) –
- **compliance_policies[].closing_event_display_type** (*string*) –
- **compliance_policies[].compliance_type** (*string*) –

- `compliance_policies[].description(string)` –
- `compliance_policies[].display_name(string)` –
- `compliance_policies[].dynamic_variability(number)` –
- `compliance_policies[].dynamic_window(string)` –
- `compliance_policies[].event_display_type(string)` –
- `compliance_policies[].identity(string)` –
- `compliance_policies[].richness_assertions[].or[](string)` –
- `compliance_policies[].time_period_seconds(string)` –
- `next_page_token(string)` –

POST /archivist/v1/compliance_policies

Request JSON Object

- `asset_filter[].or[](string)` –
- `closing_event_display_type(string)` – this is the correlated event type (read only)
- `compliance_type(string)` –
- `description(string)` – Customer description of the compliance policy. (read only)
- `display_name(string)` – display name (read only)
- `dynamic_variability(number)` – number of standard deviations - required for DYNAMIC_TOLERANCE (read only)
- `dynamic_window(string)` – valid period for policy - required for DYNAMIC_TOLERANCE (read only)
- `event_display_type(string)` – this is the target event_display_type - always required (read only)
- `richness_assertions[].or[](string)` –
- `time_period_seconds(string)` – time delta - required for SINCE and PERIOD_OUTSTANDING (read only)

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **402 Payment Required** – Returned when the user's quota of compliance policies has been reached.
- **403 Forbidden** – Returned when the user is not authorized to access the requested resource.
- **404 Not Found** – Returned when the asset with the id does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- `asset_filter[].or[](string)` –

- **closing_event_display_type**(*string*) –
- **compliance_type**(*string*) –
- **description**(*string*) –
- **display_name**(*string*) –
- **dynamic_variability**(*number*) –
- **dynamic_window**(*string*) –
- **event_display_type**(*string*) –
- **identity**(*string*) –
- **richness_assertions[]**.or[](*string*) –
- **time_period_seconds**(*string*) –

GET /archivist/v1/compliance_policies/{uuid}

Parameters

- **uuid** (*string*) – Specify the Compliance Policy UUID where *compliance_policies/{uuid}* is the Compliance Policy Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *compliance_policies/add30235-1424-4fda-840a-d5ef82c4c96f*

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to access the requested resource.
- **404 Not Found** – Returned when the asset with the id does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **asset_filter[]**.or[](*string*) –
- **closing_event_display_type**(*string*) –
- **compliance_type**(*string*) –
- **description**(*string*) –
- **display_name**(*string*) –
- **dynamic_variability**(*number*) –
- **dynamic_window**(*string*) –
- **event_display_type**(*string*) –
- **identity**(*string*) –
- **richness_assertions[]**.or[](*string*) –
- **time_period_seconds**(*string*) –

DELETE /archivist/v1/compliance_policies/{uuid}

Parameters

- **uuid** (*string*) – Specify the Compliance Policy UUID where *compliance_policies/{uuid}* is the Compliance Policy Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *compliance_policies/add30235-1424-4fda-840a-d5ef82c4c96f*

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to access the requested resource.
- **404 Not Found** – Returned when the asset with the id does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **asset_filter[]**.**or[]** (*string*) –
- **closing_event_display_type** (*string*) –
- **compliance_type** (*string*) –
- **description** (*string*) –
- **display_name** (*string*) –
- **dynamic_variability** (*number*) –
- **dynamic_window** (*string*) –
- **event_display_type** (*string*) –
- **identity** (*string*) –
- **richness_assertions[]**.**or[]** (*string*) –
- **time_period_seconds** (*string*) –

GET /archivist/v1/compliance_policies:caps
Get remaining capped resources for CompliancePolicies

Not stable or officially supported. Get remaining capped resources for CompliancePolicies

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **caps[]**.**resource_remaining** (*string*) – Number of capped resources remaining
- **caps[]**.**resource_type** (*string*) – String identifying the capped resource type

GET /archivist/v1/compliance_policies:openapi
Get OpenAPI spec for Compliance

Get OpenAPI v2.0 spec for Compliance

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

GET /archivist/v1/compliance_policies:openapi-ui
Get OpenAPI UI for Compliance

Get OpenAPI v2.0 UI for Compliance

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

IDENTITY AND ACCESS MANAGEMENT (V1)

Organisational access to the various resources in Jitsuin Archivist is managed by the subjects and access_policies endpoints.

The subjects endpoint manages the granting of access to third parties. The access_policies endpoint manages which rights have been granted.

7.1 IAM Access Policies API (v1)

7.1.1 IAM Access Policies Creation (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Define the access_policies parameters and store in /path/to/jsonfile:

```
{
  "display_name": "Friendly name of the policy",
  "description": "Description of the policy",
  "filters": [
    { "or": [
      "attributes.arc_home_location_identity=locations/5ea815f0-4de1-4a84-9377-
↪701e880fe8ae",
      "attributes.arc_home_location_identity=locations/27eed70b-9e2b-4db1-b8c4-
↪e36505350dcc"
    ]},
    { "or": [
      "attributes.arc_display_type=Valve",
      "attributes.arc_display_type=Pump"
    ]},
    { "or": [
      "attributes.ext_vendor_name=SynsationIndustries"
    ]}
  ]
}
```

(continues on next page)

(continued from previous page)

```

    ],
    "access_permissions": [
      {
        "asset_attributes_read": [ "toner_colour", "toner_type" ],
        "asset_attributes_write": [ "toner_colour" ],
        "behaviours": [ "Attachments", "Firmware", "Maintenance", "RecordEvidence"
→ " ],
        "event_arc_display_type_read": [ "toner_type", "toner_colour" ],
        "event_arc_display_type_write": [ "toner_replacement" ],
        "include_attributes": [ "arc_display_name", "arc_display_type", "arc_
→ firmware_version" ],
        "subjects": [
          "subjects/6a951b62-0a26-4c22-a886-1082297b063b",
          "subjects/a24306e5-dc06-41ba-a7d6-2b6b3e1df48d"
        ],
        "user_attributes": [
          { "or": [ "group:maintainers", "group:supervisors" ] }
        ]
      }
    ]
  }
}

```

Note:

display_name required Friendly name for the policy. Displayed in the Archivist GUI.

description Description of the policy.

filters list of filters of asset attributes to match.

access_permissions A list specifying which subjects and users get what rights for the matching assets.

behaviours list of behaviours allowed to update the asset for the matching subjects and users. For all behaviours use ["*"]

At least one of the following fields is required.

asset_attributes_read asset attributes named in this list will be visible.

asset_attributes_write asset attributes named in this list will be writable. Note they can only be read if also listed in `asset_attributes_read`.

event_arc_display_type_read events which have an event attribute `arc_display_type` with a value from this list will be visible. Matches due to `event_arc_display_type_read` are OR'd with matches due to `include_attributes`. To share all events with the specified users for any asset matching the filters, use ["*"]. Using "*" means the event can have any value in `arc_display_type` or can omit it all together.

event_arc_display_type_write events which have an event attribute `arc_display_type` with a value from this list will be WRITABLE. Matches due to `event_arc_display_type_write` are OR'd with matches due to `include_attributes`. To share all events with the specified users for any asset matching the filters, use ["*"]. Using "*" means the event can set any value in `arc_display_type` or can omit it all together.

include_attributes list of attributes to share with the matching subjects and be visible to the matching users. For all attributes use ["*"]. matches due to `include_attributes` are OR'd with `event_arc_display_type_read`

subjects list of subject identities of subjects who are to be granted these rights

user_attributes list of user attribute filters that specifies who is allowed to see the assets matching the policy filters and use those assets behaviours

Create the access policy:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/iam/v1/access_policies
```

The response is:

```
{
  "identity": "access_policies/3f5be24f-fd1b-40e2-af35-ec7c14c74d53",
  "display_name": "Friendly name of the policy",
  "description": "Description of the policy",
  "filters": [
    { "or": [
      "attributes.arc_home_location_identity=locations/5ea815f0-4de1-4a84-9377-701e880fe8ae",
      "attributes.arc_home_location_identity=locations/27eed70b-9e2b-4db1-b8c4-e36505350dcc"
    ] },
    { "or": [
      "attributes.arc_display_type=Valve",
      "attributes.arc_display_type=Pump"
    ] },
    { "or": [
      "attributes.ext_vendor_name=SynsationIndustries"
    ] }
  ],
  "access_permissions": [
    {
      "asset_attributes_read": [ "toner_colour", "toner_type" ],
      "asset_attributes_write": [ "toner_colour" ],
      "behaviours": [ "Attachments", "Firmware", "Maintenance", "RecordEvidence" ],
      "event_arc_display_type_read": [ "toner_type", "toner_colour" ],
      "event_arc_display_type_write": [ "toner_replacement" ],
      "include_attributes": [ "arc_display_name", "arc_display_type", "arc_firmware_version" ],
      "subjects": [
        "subjects/6a951b62-0a26-4c22-a886-1082297b063b",
        "subjects/a24306e5-dc06-41ba-a7d6-2b6b3e1df48d"
      ],
      "user_attributes": [
        { "or": [ "group:maintainers", "group:supervisors" ] }
      ]
    }
  ]
}
```

Note: A full API reference is available in [Swagger POST API](#)

7.1.2 IAM Access Policies Retrieval (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://symsation.1234-5678.nodes.archivist.jitsuin.io
```

IAM access policy records in Jitsuin Archivist are tokenized at creation time and referred to in all API calls and smart contracts throughout the system by a unique identity of the form:

```
access_policies/12345678-90ab-cdef-1234-567890abcdef.
```

If you do not know the access_policy's identity you can fetch IAM access policy records using other information you do know, such as the access_policy's name.

Fetch all IAM access_policies (v1)

To fetch all IAM access_policies records, simply GET the iam/access_policies resource:

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/iam/v1/access_policies
```

Fetch specific IAM access Policy by identity (v1)

If you know the unique identity of the IAM access policy Record simply GET the resource:

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/iam/v1/access_policies/6a951b62-0a26-4c22-a886-1082297b063b
```

Fetch IAM Access Policies by name (v1)

To fetch all IAM access_policies with a specific name, GET the iam/access_policies resource and filter on display_name:

```
$ curl -g -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/iam/v1/access_policies?display_name=Some%20description
```

Each of these calls returns a list of matching IAM access_policies records in the form:

```
{
  "access_policies": [
    {
      "identity": "access_policies/6a951b62-0a26-4c22-a886-1082297b063b",
```

(continues on next page)

(continued from previous page)

```

    "display_name": "Name to display",
    "description": "Description of the policy",
    "filters": [
      { "or": [
        "attributes.arc_home_location_identity=locations/5ea815f0-4de1-
↪4a84-9377-701e880fe8ae",
        "attributes.arc_home_location_identity=locations/27eed70b-9e2b-
↪4db1-b8c4-e36505350dcc"
      ] },
      { "or": [
        "attributes.arc_display_type=Valve",
        "attributes.arc_display_type=Pump"
      ] },
      { "or": [
        "attributes.ext_vendor_name=SynsationIndustries"
      ] }
    ],
    "access_permissions": [
      {
        "asset_attributes_read": [ "toner_colour", "toner_type" ],
        "asset_attributes_write": [ "toner_colour" ],
        "behaviours": [ "Attachments", "Firmware", "Maintenance",
↪"RecordEvidence" ],
        "event_arc_display_type_read": [ "toner_type", "toner_colour" ],
        "event_arc_display_type_write": [ "toner_replacement" ],
        "include_attributes": [ "arc_display_name", "arc_display_type",
↪"arc_firmware_version" ],
        "subjects": [
          "subjects/6a951b62-0a26-4c22-a886-1082297b063b",
          "subjects/a24306e5-dc06-41ba-a7d6-2b6b3e1df48d"
        ],
        "user_attributes": [
          { "or": [ "group:maintainers", "group:supervisors" ] }
        ]
      }
    ]
  },
  {
    "identity": "access_policies/12345678-0a26-4c22-a886-1082297b063b",
    "display_name": "Some other description",
    "filters": [
      { "or": [ "attributes.arc_display_type=door_access_reader" ] }
    ],
    "access_permissions": [
      {
        "asset_attributes_read": [ "toner_colour", "toner_type" ],
        "asset_attributes_write": [ "toner_colour" ],
        "behaviours": [ "Attachments", "Firmware", "Maintenance",
↪"RecordEvidence" ],
        "event_arc_display_type_read": [ "toner_type", "toner_colour" ],
        "event_arc_display_type_write": [ "toner_replacement" ],
        "include_attributes": [ "arc_display_name", "arc_display_type",
↪"arc_firmware_version" ],
        "subjects": [
          "subjects/6a951b62-0a26-4c22-a886-1082297b063b",
          "subjects/a24306e5-dc06-41ba-a7d6-2b6b3e1df48d"
        ]
      }
    ]
  }
]

```

(continues on next page)

(continued from previous page)

```
        "user_attributes": [
            {
                "or": [
                    "group:maintainers",
                    "group:supervisors"
                ]
            }
        ]
    }
}
```

Note: The number of records returned has a maximum limit. If this limit is too small then one must use [API Request Paging](#).

Note: The total number of assets that exist is returned in the response header field ‘x-total-count’ if the ‘x-request-total-count’ header on the request is set to ‘true’. The curl option ‘-i’ will emit this to stdout.

Note: A full API reference is available in [Swagger GET API](#)

7.1.3 IAM Access Policy Deletion (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See [API Request Authorization and Authentication](#).

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

To delete an IAM access policy, issue following request:

```
$ curl -v -X DELETE \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  $URL/archivist/iam/v1/access_policies/47b58286-ff0f-11e9-8f0b-362b9e155667
```

The response is:

```
{ }
```

Note: A full API reference is available in [Swagger DELETE API](#)

7.1.4 IAM Access Policies Update (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Define the access_policies parameters to be changed and store in /path/to/jsonfile:

```
{
  "filters": [
    { "or": [
      "attributes.arc_home_location_identity=locations/5ea815f0-4de1-4a84-9377-701e880fe8ae",
      "attributes.arc_home_location_identity=locations/27eed70b-9e2b-4db1-b8c4-e36505350dcc"
    ] },
    { "or": [
      "attributes.arc_display_type=Valve",
      "attributes.arc_display_type=Pump"
    ] },
    { "or": [
      "attributes.ext_vendor_name=SynsationIndustries"
    ] }
  ],
  "access_permissions": [
    {
      "asset_attributes_read": [ "toner_colour", "toner_type" ],
      "asset_attributes_write": [ "toner_colour" ],
      "behaviours": [ "Attachments", "Firmware", "Maintenance", "RecordEvidence" ],
      "event_arc_display_type_read": [ "toner_type", "toner_colour" ],
      "event_arc_display_type_write": [ "toner_replacement" ],
      "include_attributes": [ "arc_display_name", "arc_display_type", "arc_firmware_version" ],
      "subjects": [
        "subjects/6a951b62-0a26-4c22-a886-1082297b063b",
        "subjects/a24306e5-dc06-41ba-a7d6-2b6b3e1df48d"
      ],
      "user_attributes": [
        { "or": [ "group:maintainers", "group:supervisors" ] }
      ]
    }
  ]
}
```

Note:

filters list of asset attributes filters.

access_permissions A list specifying which subjects and users get what rights for the matching assets.

behaviours list of behaviours allowed to update the asset for the matching subjects and users. For all behaviours use ["*"]

asset_attributes_read asset attributes named in this list will be visible.

asset_attributes_write asset attributes named in this list will be writable. Note they can only be read if also listed in `asset_attributes_read`.

event_arc_display_type_read events which have an event attribute `arc_display_type` with a value from this list will be visible. Matches due to `event_arc_display_type_read` are OR'd with matches due to `include_attributes`. To share all events with the specified users for any asset matching the filters, use ["*"]. Using "*" means the event can have any value in `arc_display_type` or can omit it all together.

event_arc_display_type_write events which have an event attribute `arc_display_type` with a value from this list will be WRITABLE. Matches due to `event_arc_display_type_write` are OR'd with matches due to `include_attributes`. To share all events with the specified users for any asset matching the filters, use ["*"]. Using "*" means the event can set any value in `arc_display_type` or can omit it all together.

include_attributes list of attributes to share with the matching subjects and be visible to the matching users. For all attributes use ["*"]. matches due to `include_attributes` are OR'd with `event_arc_display_type_read`

subjects list of subject identities of subjects who are to be granted these rights

user_attributes list of user attribute filters that specifies who is allowed to see the assets matching the policy filters and use those assets behaviours

Update the access policy:

```
$ curl -v -X PATCH \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@path/to/jsonfile" \
  $URL/archivist/iam/v1/access_policies/47b58286-ff0f-11e9-8f0b-362b9e155667
```

The response is:

```
{
  "identity": "access_policies/3f5be24f-fd1b-40e2-af35-ec7c14c74d53",
  "display_name": "Friendly name of the policy",
  "description": "Description of the policy",
  "filters": [
    { "or": [
      "attributes.arc_home_location_identity=locations/5ea815f0-4de1-4a84-9377-701e880fe8ae",
      "attributes.arc_home_location_identity=locations/27eed70b-9e2b-4db1-b8c4-e36505350dcc"
    ] },
    { "or": [
      "attributes.arc_display_type=Valve",
      "attributes.arc_display_type=Pump"
    ] },
    { "or": [
      "attributes.ext_vendor_name=SynsationIndustries"
    ] }
  ],
  "access_permissions": [
    {
      "asset_attributes_read": [ "toner_colour", "toner_type" ],
      "asset_attributes_write": [ "toner_colour" ],
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    "behaviours": [ "Attachments", "Firmware", "Maintenance", "RecordEvidence
    ↪ " ],
    "event_arc_display_type_read": [ "toner_type", "toner_colour"],
    "event_arc_display_type_write": [ "toner_replacement"],
    "include_attributes": [ "arc_display_name", "arc_display_type", "arc_
    ↪ firmware_version" ],
    "subjects": [
        "subjects/6a951b62-0a26-4c22-a886-1082297b063b",
        "subjects/a24306e5-dc06-41ba-a7d6-2b6b3e1df48d"
    ],
    "user_attributes": [
        { "or": [ "group:maintainers", "group:supervisors" ] }
    ]
  }
}

```

Note: A full API reference is available in [Swagger PATCH API](#)

7.1.5 IAM Access Policies matching Assets (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://symsation.1234-5678.nodes.archivist.jitsuin.io
```

IAM access policy records in Jitsuin Archivist are tokenized at creation time and referred to in all API calls and smart contracts throughout the system by a unique identity of the form:

```
access_policies/12345678-90ab-cdef-1234-567890abcdef.
```

If you do not know the access_policy's identity you can fetch IAM access policy records using other information you do know, such as the access_policy's name.

Fetch all Assets matching specific IAM access_policy (v1)

If you know the unique identity of the IAM access policy Record simply GET the resource:

```
$ curl -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/iam/v1/access_policies/6a951b62-0a26-4c22-a886-1082297b063b/assets
```

Each of these calls returns a list of matching Asset records in the form:

```
{
  "assets": [
    {
      "identity": "assets/6a951b62-0a26-4c22-a886-1082297b063b",
      "behaviours": [
        "Firmware",
        "Maintenance",
        "RecordEvidence",
        "LocationUpdate",
        "Attachments"
      ],
      "attributes": {
        "arc_display_type": "Pump",
        "arc_firmware_version": "1.0",
        "arc_home_location_identity": "locations/866790d8-4ed6-4cc9-8f60-
↪07672609b331",
        "arc_serial_number": "vtl-x4-07",
        "arc_description": "Pump at A603 North East",
        "arc_display_name": "tcl.ccj.003",
        "some_custom_attribute": "value",
        "arc_attachments": [
          {
            "arc_display_name": "arc_primary_image",
            "arc_attachment_identity": "blobs/87b1a84c-1c6f-442b-923e-
↪a97516f4d275",
            "arc_hash_alg": "SHA256",
            "arc_hash_value":
↪"246c316e2cd6971ce5c83a3e61f9880fa6e2f14ae2976ee03500eb282fd03a60"
          }
        ]
      },
      "confirmation_status": "CONFIRMED",
      "tracked": "TRACKED"
    }
  ]
}
```

Fetch all IAM access_policies matching specific Asset (v1)

If you know the unique identity of the Asset Record simply GET matching policies:

```
$ curl -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/iam/v1/assets/6a951b62-0a26-4c22-a886-1082297b063b/access_policies
```

Each of these calls returns a list of matching IAM access_policies records in the form:

```
{
  "access_policies": [
    {
      "identity": "access_policies/6a951b62-0a26-4c22-a886-1082297b063b",
      "display_name": "Some description",
      "filters": [
        { "or": [
            "attributes.arc_home_location_identity=locations/5ea815f0-4de1-
↪4a84-9377-701e880fe8ae",

```

(continues on next page)

(continued from previous page)

```

        "attributes.arc_home_location_identity=locations/27eed70b-9e2b-
↪4db1-b8c4-e36505350dcc",
        ],
        { "or": [
            "attributes.arc_display_type=Valve",
            "attributes.arc_display_type=Pump"
        ]},
        { "or": [
            "attributes.ext_vendor_name=SynsationIndustries"
        ]}
    ],
    "access_permissions": [
        {
            "subjects": [
                "subjects/6a951b62-0a26-4c22-a886-1082297b063b",
                "subjects/a24306e5-dc06-41ba-a7d6-2b6b3e1df48d"
            ],
            "behaviours": [ "Attachments", "Firmware", "Maintenance",
↪"RecordEvidence" ],
            "include_attributes": [ "arc_display_name", "arc_display_type",
↪"arc_firmware_version" ],
            "user_attributes": [
                { "or": ["group:maintainers", "group:supervisors"]}
            ]
        }
    ]
},
{
    "identity": "access_policies/12345678-0a26-4c22-a886-1082297b063b",
    "display_name": "Some other description",
    "filters": [
        { "or": ["attributes.arc_display_type=door_access_reader"]}
    ],
    "access_permissions": [
        {
            "subjects": [
                "subjects/6a951b62-0a26-4c22-a886-1082297b063b",
                "subjects/a24306e5-dc06-41ba-a7d6-2b6b3e1df48d"
            ],
            "behaviours": [ "Attachments", "Maintenance", "RecordEvidence" ],
            "include_attributes": [ "arc_display_name", "arc_display_type" ],
            "user_attributes": [
                { "or": ["group:maintainers", "group:supervisors"]}
            ]
        }
    ]
}
]
}
}

```

Note: The number of records returned has a maximum limit. If this limit is too small then one must use *API Request Paging*.

A full API reference is available in [Swagger GET API](#)

7.1.6 IAM access Policies Swagger API

GET /archivist/iam/v1/access_policies
List access policies

Returns a paginated list of access_policies

Query Parameters

- **order_by** (*string*) – Specify the sort order for the results.
- **page_size** (*integer*) – Maximum entries per page
- **page_token** (*string*) – The next_page_token returned from a previous list request if any.
- **display_name** (*string*) – Further fields are bound to query parameters and act to filter the result
Customer friendly name for the access policy.
- **description** (*string*) – Customer description of the access policy.

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to list the access policy.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **access_policies[] .access_permissions[] .asset_attributes_read[]** (*string*) –
- **access_policies[] .access_permissions[] .asset_attributes_write[]** (*string*) –
- **access_policies[] .access_permissions[] .behaviours[]** (*string*) –
- **access_policies[] .access_permissions[] .event_arc_display_type_read[]** (*string*) –
- **access_policies[] .access_permissions[] .event_arc_display_type_write[]** (*string*) –
- **access_policies[] .access_permissions[] .include_attributes[]** (*string*) –
- **access_policies[] .access_permissions[] .subjects[]** (*string*) –
- **access_policies[] .access_permissions[] .user_attributes[] .or[]** (*string*) –
- **access_policies[] .description** (*string*) – Customer description for the access policy.

- **access_policies[] .display_name** (*string*) – Customer friendly name for the access policy.
- **access_policies[] .filters[] .or[]** (*string*) –
- **access_policies[] .identity** (*string*) – Unique identification for the access policy, Relative Resource Name
- **access_policies[] .tenant** (*string*) – Tenant id
- **next_page_token** (*string*) – Token to retrieve the next page of results or empty if there are none.

POST /archivist/iam/v1/access_policies

Create an access policy

This request creates a new access policy. The display_name is the friendly name.

Request JSON Object

- **access_permissions[] .asset_attributes_read[]** (*string*) –
- **access_permissions[] .asset_attributes_write[]** (*string*) –
- **access_permissions[] .behaviours[]** (*string*) –
- **access_permissions[] .event_arc_display_type_read[]** (*string*) –
- **access_permissions[] .event_arc_display_type_write[]** (*string*) –
- **access_permissions[] .include_attributes[]** (*string*) –
- **access_permissions[] .subjects[]** (*string*) –
- **access_permissions[] .user_attributes[] .or[]** (*string*) –
- **description** (*string*) – Customer description for the access policy.
- **display_name** (*string*) – Customer friendly name for the access policy. (required)
- **filters[] .or[]** (*string*) –

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **402 Payment Required** – Returned when the user's quota of access policies has been reached.
- **403 Forbidden** – Returned when the user is not authorized to create an access policy.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **access_permissions[] .asset_attributes_read[]** (*string*) –
- **access_permissions[] .asset_attributes_write[]** (*string*) –
- **access_permissions[] .behaviours[]** (*string*) –
- **access_permissions[] .event_arc_display_type_read[]** (*string*) –

- **access_permissions[] .event_arc_display_type_write[]** (*string*) –
- **access_permissions[] .include_attributes[]** (*string*) –
- **access_permissions[] .subjects[]** (*string*) –
- **access_permissions[] .user_attributes[] .or[]** (*string*) –
- **description** (*string*) – Customer description for the access policy.
- **display_name** (*string*) – Customer friendly name for the access policy.
- **filters[] .or[]** (*string*) –
- **identity** (*string*) – Unique identification for the access policy, Relative Resource Name
- **tenant** (*string*) – Tenant id

GET /archivist/iam/v1/access_policies/{uuid}

Get an access policy

Returns the identified access policy

Parameters

- **uuid** (*string*) – Specify the Access Policy UUID where *access_policies/{uuid}* is the Access Policy Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *access_policies/add30235-1424-4fda-840a-d5ef82c4c96f*

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to read the access policy.
- **404 Not Found** – Returned when the identified access policy does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **access_permissions[] .asset_attributes_read[]** (*string*) –
- **access_permissions[] .asset_attributes_write[]** (*string*) –
- **access_permissions[] .behaviours[]** (*string*) –
- **access_permissions[] .event_arc_display_type_read[]** (*string*) –
- **access_permissions[] .event_arc_display_type_write[]** (*string*) –
- **access_permissions[] .include_attributes[]** (*string*) –
- **access_permissions[] .subjects[]** (*string*) –
- **access_permissions[] .user_attributes[] .or[]** (*string*) –
- **description** (*string*) – Customer description for the access policy.
- **display_name** (*string*) – Customer friendly name for the access policy.

- **filters[]**.or[] (*string*) –
- **identity** (*string*) – Unique identification for the access policy, Relative Resource Name
- **tenant** (*string*) – Tenant id

DELETE /archivist/iam/v1/access_policies/{uuid}**Delete an access policy**

Delete the identified access policy

Parameters

- **uuid** (*string*) – Specify the Access Policy UUID where *access_policies/{uuid}* is the Access Policy Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *access_policies/add30235-1424-4fda-840a-d5ef82c4c96f*

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to delete the access policy.
- **404 Not Found** – Returned when the identified access policy does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

PATCH /archivist/iam/v1/access_policies/{uuid}**Update a access policy details**

Perform a full or partial update of the identified access policy

Parameters

- **uuid** (*string*) – Specify the Access Policy UUID where *access_policies/{uuid}* is the Access Policy Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *access_policies/add30235-1424-4fda-840a-d5ef82c4c96f*

Query Parameters

- **mask** (*string*) – The mask says exactly which fields are to be updated. This removes the ambiguity caused by 'zero' valued fields. <https://grpc-ecosystem.github.io/grpc-gateway/docs/patch.html>

Request JSON Object

- **access_permissions[]**.asset_attributes_read[] (*string*) –
- **access_permissions[]**.asset_attributes_write[] (*string*) –
- **access_permissions[]**.behaviours[] (*string*) –
- **access_permissions[]**.event_arc_display_type_read[] (*string*) –
- **access_permissions[]**.event_arc_display_type_write[] (*string*) –
- **access_permissions[]**.include_attributes[] (*string*) –

- **access_permissions[] .subjects[]** (*string*) –
- **access_permissions[] .user_attributes[] .or[]** (*string*) –
- **description** (*string*) – Customer description for the access policy.
- **display_name** (*string*) – Customer friendly name for the access policy.
- **filters[] .or[]** (*string*) –
- **identity** (*string*) – Unique identification for the access policy, Relative Resource Name
- **tenant** (*string*) – Tenant id

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to update the access policy.
- **404 Not Found** – Returned when the identified access policy does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **access_permissions[] .asset_attributes_read[]** (*string*) –
- **access_permissions[] .asset_attributes_write[]** (*string*) –
- **access_permissions[] .behaviours[]** (*string*) –
- **access_permissions[] .event_arc_display_type_read[]** (*string*) –
- **access_permissions[] .event_arc_display_type_write[]** (*string*) –
- **access_permissions[] .include_attributes[]** (*string*) –
- **access_permissions[] .subjects[]** (*string*) –
- **access_permissions[] .user_attributes[] .or[]** (*string*) –
- **description** (*string*) – Customer description for the access policy.
- **display_name** (*string*) – Customer friendly name for the access policy.
- **filters[] .or[]** (*string*) –
- **identity** (*string*) – Unique identification for the access policy, Relative Resource Name
- **tenant** (*string*) – Tenant id

GET /archivist/iam/v1/access_policies/{uuid}/assets
Returns assets matching access policy

Returns assets matching access policy

Parameters

- **uuid** (*string*) – Specify the Access Policy UUID where *access_policies/{uuid}* is the Access Policy Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *access_policies/add30235-1424-4fda-840a-d5ef82c4c96f*

Query Parameters

- **order_by** (*string*) – Specify the sort order for the results.
- **page_size** (*integer*) – Maximum entries per page
- **page_token** (*string*) – The next_page_token returned from a previous list request if any.

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to list the access policy.
- **404 Not Found** – Returned when the identified access policy does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **assets[] .access_policy** (*object*) – key value mapping of behaviour to private for keys
- **assets[] .at_time** (*string*) – indicates time the asset data is from (read only)
- **assets[] .attributes** (*object*) – key value mapping of asset properties
- **assets[] .behaviours[]** (*string*) –
- **assets[] .chain_id** (*string*) – chain id of the blockchain associated with this asset (read only)
- **assets[] .confirmation_status** (*string*) –
- **assets[] .identity** (*string*) – relative resource address *assets/{UUID}* (read only)
- **assets[] .owner** (*string*) – wallet address of the asset owner (read only)
- **assets[] .proof_mechanism** (*string*) –
- **assets[] .public** (*boolean*) – Public asset
- **assets[] .storage_integrity** (*string*) – Specifies how the asset data will be stored. This is set once on creation and does not change.
- **assets[] .tenant_identity** (*string*) – Identity of the tenant the that created this asset
- **assets[] .tracked** (*string*) –
- **next_page_token** (*string*) – Token to retrieve the next page of results or empty if there are none.

GET /archivist/iam/v1/access_policies:caps
Get remaining capped resources for AccessPolicies

Not stable or officially supported. Get remaining capped resources for AccessPolicies

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **caps[] .resource_remaining** (*string*) – Number of capped resources remaining
- **caps[] .resource_type** (*string*) – String identifying the capped resource type

GET /archivist/iam/v1/assets/{uuid}/access_policies
Get matching access policies

Get matching access policies for specified asset

Parameters

- **uuid** (*string*) – Specify the Asset UUID where *assets/{uuid}* is the Asset Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *assets/add30235-1424-4fda-840a-d5ef82c4c96f*

Query Parameters

- **at_time** (*string*) – Specify time in the past to show asset data as it was at time specified

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to list the access policy.
- **404 Not Found** – Returned when the identified access policy does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **access_policies[] .access_permissions[] .asset_attributes_read[]** (*string*) –
- **access_policies[] .access_permissions[] .asset_attributes_write[]** (*string*) –
- **access_policies[] .access_permissions[] .behaviours[]** (*string*) –
- **access_policies[] .access_permissions[] .event_arc_display_type_read[]** (*string*) –

- **access_policies[] .access_permissions[] .event_arc_display_type_write[]** (*string*) –
- **access_policies[] .access_permissions[] .include_attributes[]** (*string*) –
- **access_policies[] .access_permissions[] .subjects[]** (*string*) –
- **access_policies[] .access_permissions[] .user_attributes[] .or[]** (*string*) –
- **access_policies[] .description** (*string*) – Customer description for the access policy.
- **access_policies[] .display_name** (*string*) – Customer friendly name for the access policy.
- **access_policies[] .filters[] .or[]** (*string*) –
- **access_policies[] .identity** (*string*) – Unique identification for the access policy, Relative Resource Name
- **access_policies[] .tenant** (*string*) – Tenant id
- **next_page_token** (*string*) – Token to retrieve the next page of results or empty if there are none.

GET /archivist/iam/v1/access_policies:openapi
Get OpenAPI spec for AccessPolicies

Get OpenAPI v2.0 spec for AccessPolicies

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

GET /archivist/iam/v1/access_policies:openapi-ui
Get OpenAPI UI for AccessPolicies

Get OpenAPI v2.0 UI for AccessPolicies

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

7.2 IAM Subjects API (v1)

7.2.1 IAM Subjects Creation (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Define the subjects parameters and store in /path/to/jsonfile:

```
{
  "display_name": "Some description",
  "wallet_pub_key": ["key1"],
  "tessera_pub_key": ["key2"]
}
```

Note:

display_name required Friendly name for the location. Displayed in the Archivist GUI.

wallet_pub_key required a list containing a single organisation wallet key.

tessera_pub_key required a list containing the single tessera key for the archivist node the subject has residency on

Create the IAM subject:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/iam/v1/subjects
```

The response is:

```
{
  "identity": "subjects/3f5be24f-fd1b-40e2-af35-ec7c14c74d53",
  "display_name": "Some description",
  "wallet_pub_key": ["key1"],
  "wallet_address": ["address"],
  "tessera_pub_key": ["key2"]
}
```

Note: A full API reference is available in [Swagger POST API](#)

7.2.2 IAM Subjects Retrieval (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

IAM subject records in Jitsuin Archivist are tokenized at creation time and referred to in all API calls and smart contracts throughout the system by a unique identity of the form:

```
subjects/12345678-90ab-cdef-1234-567890abcdef.
```

If you do not know the subjects's identity you can fetch IAM subject records using other information you do know, such as the subject's name.

Fetch all IAM subjects (v1)

To fetch all IAM subjects records, simply GET the `/subjects` resource:

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/iam/v1/subjects
```

Fetch specific IAM Subject by identity (v1)

If you know the unique identity of the IAM subject Record simply GET the resource:

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/iam/v1/subjects/6a951b62-0a26-4c22-a886-1082297b063b
```

Fetch IAM Subjects by name (v1)

To fetch all IAM subjects with a specific name, GET the `/subjects` resource and filter on `display_name`:

```
$ curl -g -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/iam/v1/subjects?display_name=Acme
```

Each of these calls returns a list of matching IAM subjects records in the form:

```
{
  "subjects": [
    {
      "identity": "subjects/6a951b62-0a26-4c22-a886-1082297b063b",
      "display_name": "Some description",

```

(continues on next page)

(continued from previous page)

```
    "wallet_pub_key": ["key1"],
    "wallet_address": ["address1"],
    "tessera_pub_key": ["key2"]
  },
  {
    "identity": "subjects/12345678-0a26-4c22-a886-1082297b063b",
    "display_name": "Some otherdescription",
    "wallet_pub_key": ["key5"],
    "wallet_address": ["address5"],
    "tessera_pub_key": ["key7"]
  }
]
```

Note: The number of records returned has a maximum limit. If this limit is too small then one must use [API Request Paging](#).

Note: The total number of subjects that exist is returned in the response header field 'x-total-count' if the 'x-request-total-count' header on the request is set to 'true'. The curl option '-i' will emit this to stdout.

Note: A full API reference is available in [Swagger GET API](#)

7.2.3 IAM Subject Deletion (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See [API Request Authorization and Authentication](#).

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

To delete an IAM subject, issue following request:

```
$ curl -v -X DELETE \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  $URL/archivist/iam/v1/subjects/47b58286-ff0f-11e9-8f0b-362b9e155667
```

The response is:

```
{}
```

Note: A full API reference is available in [Swagger DELETE API](#)

7.2.4 IAM Subjects Update (v1)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://symsation.1234-5678.nodes.archivist.jitsuin.io
```

Define the subjects parameters to be changed and store in /path/to/jsonfile:

```
{
  "wallet_pub_key": ["key1"],
  "tessera_pub_key": ["key2"]
}
```

Note:

wallet_pub_key list of organisation wallet keys

tessera_pub_key list of organisation tessera keys

Update the IAM Subject:

```
$ curl -v -X PATCH \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/iam/v1/subjects/47b58286-ff0f-11e9-8f0b-362b9e155667
```

The response is:

```
{
  "identity": "subjects/3f5be24f-fd1b-40e2-af35-ec7c14c74d53",
  "display_name": "Some description",
  "wallet_pub_key": ["key1"],
  "wallet_address": ["address1"],
  "tessera_pub_key": ["key3"]
}
```

Note: A full API reference is available in [Swagger PATCH API](#)

7.2.5 IAM Subjects Self Entry (v1)

There is an immutable entry in the subjects API called “Self” that contains the keys for the hosting organisation of the Jitsuin Archivist Node.

This entry cannot be deleted or updated.

This special identity is:

```
subjects/00000000-0000-0000-0000-000000000000
```

Fetch self IAM Subject by identity (v1)

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/iam/v1/subjects/00000000-0000-0000-0000-000000000000
```

Response

```
[
  {
    "identity": "subjects/00000000-0000-0000-0000-000000000000",
    "display_name": "Some description",
    "wallet_pub_key": ["key1"],
    "wallet_address": ["address1"],
    "tesseract_pub_key": ["key3"]
  }
]
```

Note: A full API reference is available in [Swagger GET API](#)

7.2.6 IAM Subjects Swagger API

GET /archivist/iam/v1/subjects

List subjects

Returns a paginated list of subjects

Query Parameters

- **order_by** (*string*) – Specify the sort order for the results.
- **page_size** (*integer*) – Maximum entries per page
- **page_token** (*string*) – The next_page_token returned from a previous list request if any.
- **display_name** (*string*) – Display name for filtering
Customer friendly name for the subject.
- **wallet_address** (*string*) – XXX: Investigate max length. Its 256 elsewhere in this file, but that doesn't seem right. Wallet address for filtering
Customer friendly name for the subject.

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to read the subject.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **next_page_token** (*string*) – Token to retrieve the next page of results or empty if there are none.
- **subjects[] .confirmation_status** (*string*) –
- **subjects[] .display_name** (*string*) – Customer friendly name for the subject.
- **subjects[] .identity** (*string*) – Unique identification for the subject, Relative Resource Name
- **subjects[] .tenant** (*string*) – Tenant id
- **subjects[] .tesseract_pub_key[]** (*string*) –
- **subjects[] .wallet_address[]** (*string*) –
- **subjects[] .wallet_pub_key[]** (*string*) –

POST /archivist/iam/v1/subjects Create an subject

This request creates a new subject. The display_name is the friendly name.

Request JSON Object

- **display_name** (*string*) – Customer friendly name for the subject. (required)
- **tesseract_pub_key[]** (*string*) –
- **wallet_pub_key[]** (*string*) –

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to create a subject.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **confirmation_status** (*string*) –
- **display_name** (*string*) – Customer friendly name for the subject.

- **identity** (*string*) – Unique identification for the subject, Relative Resource Name
- **tenant** (*string*) – Tenant id
- **tessera_pub_key[]** (*string*) –
- **wallet_address[]** (*string*) –
- **wallet_pub_key[]** (*string*) –

GET /archivist/iam/v1/subjects/{uuid}
Get an subject

Returns the identified subject

Parameters

- **uuid** (*string*) – Specify the Subject UUID where *subjects/{uuid}* is the Subject Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *subjects/add30235-1424-4fda-840a-d5ef82c4c96f*. Note the special subject *00000000-0000-0000-0000-000000000000* represents the *self* subject, or your own organisation.

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to read the subject.
- **404 Not Found** – Returned when the identified subject does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **confirmation_status** (*string*) –
- **display_name** (*string*) – Customer friendly name for the subject.
- **identity** (*string*) – Unique identification for the subject, Relative Resource Name
- **tenant** (*string*) – Tenant id
- **tessera_pub_key[]** (*string*) –
- **wallet_address[]** (*string*) –
- **wallet_pub_key[]** (*string*) –

DELETE /archivist/iam/v1/subjects/{uuid}
Delete a subject

Delete the identified subject

Parameters

- **uuid** (*string*) – Specify the Subject UUID where *subjects/{uuid}* is the Subject Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *subjects/add30235-1424-4fda-840a-d5ef82c4c96f*

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to delete the subject.
- **404 Not Found** – Returned when the identified laccess policy does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription’s rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

PATCH /archivist/iam/v1/subjects/{uuid}

Update a subject’s details

Perform a full or partial update of the identified subject

Parameters

- **uuid** (*string*) – Specify the Subject UUID where *subjects/{uuid}* is the Subject Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *subjects/add30235-1424-4fda-840a-d5ef82c4c96f*

Query Parameters

- **mask** (*string*) – The mask says exactly which fields are to be updated. This removes the ambiguity caused by ‘zero’ valued fields. <https://grpc-ecosystem.github.io/grpc-gateway/docs/patch.html>

Request JSON Object

- **display_name** (*string*) – Customer friendly name for the subject.
- **identity** (*string*) – Unique identification for the subject, Relative Resource Name
- **tenant** (*string*) – Tenent id
- **tessera_pub_key[]** (*string*) –
- **wallet_address[]** (*string*) –
- **wallet_pub_key[]** (*string*) –

Status Codes

- **200 OK** – A successful response.
- **400 Bad Request** – Returned when the request is badly formed.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to update the subject.
- **404 Not Found** – Returned when the identified subject does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription’s rate limit for requests.
- **500 Internal Server Error** – Returned when the underlying storage system returns an error.
- **default** – An unexpected error response.

Response JSON Object

- **confirmation_status** (*string*) –
- **display_name** (*string*) – Customer friendly name for the subject.
- **identity** (*string*) – Unique identification for the subject, Relative Resource Name
- **tenant** (*string*) – Tenant id
- **tessera_pub_key[]** (*string*) –
- **wallet_address[]** (*string*) –
- **wallet_pub_key[]** (*string*) –

GET /archivist/iam/v1/subjects:openapi
Get OpenAPI spec for Subjects

Get OpenAPI v2.0 spec for Subjects

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

GET /archivist/iam/v1/subjects:openapi-ui
Get OpenAPI UI for Subjects

Get OpenAPI v2.0 UI for Subjects

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

ATTACHMENTS API

Warning: This endpoint is currently deprecated and has been replaced by the Blobs endpoint.

8.1 Retrieve Attachment

Warning: This endpoint is currently deprecated and has been replaced by the Blobs endpoint.

Note: Attachments in the JitsuIn Archivist system are not first-order objects in their own right: they are properties of other objects such as Asset Records or events. Due to this, Attachments **MUST** be retrieved by full unique resource identity as stored in an asset Record or Event property. They cannot be listed, filtered, or searched.

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Retrieve a specific Attachment

```
$ curl -v \
  -H "@$BEARER_TOKEN_FILE" \
  -H "content_type=image/jpeg" \
  -F "file=@/path/to/file" \
  $URL/archivist/v1/attachments/08838336-c357-460d-902a-3aba9528dd22
```

The response is:

```
{
  "identity": "attachments/08838336-c357-460d-902a-3aba9528dd22",
  "hash": {
    "alg": "SHA256",
```

(continues on next page)

(continued from previous page)

```

    "value": "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  },
  "mime_type": "image/jpeg",
  "timestamp_accepted": "2019-11-07T15:31:49Z",
  "size": 31424
}

```

Note: The number of records returned has a maximum limit. If this limit is too small then one must use [API Request Paging](#).

A full API reference is available in [Swagger GET API](#)

8.2 Attachments Swagger API

GET /archivist/v1/attachments/{uuid}

Get an attachment

Returns the attachment associated with the relative resource name

Parameters

- **uuid** (*string*) – Specify the Attachment UUID where *attachments/{uuid}* is the Attachment Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *attachments/add30235-1424-4fda-840a-d5ef82c4c96f*

Status Codes

- **200 OK** – A successful response.(streaming responses)
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to read the attachment.
- **404 Not Found** – Returned when an attachment with the identity does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription’s rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **error.code** (*integer*) –
- **error.details[].@type** (*string*) – A URL/resource name that uniquely identifies the type of the serialized protocol buffer message. This string must contain at least one “/” character. The last segment of the URL’s path must represent the fully qualified name of the type (as in *path/google.protobuf.Duration*). The name should be in a canonical form (e.g., leading “.” is not accepted).

In practice, teams usually precompile into the binary all types that they expect it to use in the context of Any. However, for URLs which use the scheme *http*, *https*, or no scheme, one can optionally set up a type server that maps type URLs to message definitions as follows:

* If no scheme is provided, *https* is assumed. * An HTTP GET on the URL must yield a *[google.protobuf.Type][]* value in binary format, or produce an error. * Applications are allowed to cache lookup results based on the URL, or have them precompiled into a binary

to avoid any lookup. Therefore, binary compatibility needs to be preserved on changes to types. (Use versioned type names to manage breaking changes.)

Note: this functionality is not currently available in the official protobuf release, and it is not used for type URLs beginning with `type.googleapis.com`.

Schemes other than `http`, `https` (or the empty scheme) might be used with implementation specific semantics.

- **`error.message`** (*string*) –
- **`result.content_type`** (*string*) – The HTTP Content-Type string representing the content type of the body.
- **`result.data`** (*string*) – HTTP body binary data.
- **`result.extensions[].@type`** (*string*) – A URL/resource name that uniquely identifies the type of the serialized protocol buffer message. This string must contain at least one “/” character. The last segment of the URL’s path must represent the fully qualified name of the type (as in *path/google.protobuf.Duration*). The name should be in a canonical form (e.g., leading “.” is not accepted).

In practice, teams usually precompile into the binary all types that they expect it to use in the context of Any. However, for URLs which use the scheme `http`, `https`, or no scheme, one can optionally set up a type server that maps type URLs to message definitions as follows:

* If no scheme is provided, `https` is assumed. * An HTTP GET on the URL must yield a `[google.protobuf.Type][]` value in binary format, or produce an error. * Applications are allowed to cache lookup results based on the URL, or have them precompiled into a binary to avoid any lookup. Therefore, binary compatibility needs to be preserved on changes to types. (Use versioned type names to manage breaking changes.)

Note: this functionality is not currently available in the official protobuf release, and it is not used for type URLs beginning with `type.googleapis.com`.

Schemes other than `http`, `https` (or the empty scheme) might be used with implementation specific semantics.

GET /archivist/v1/attachments/{uuid}/info **Get attachment metadata**

Gets the metadata associated an attachment uuid

Parameters

- **`uuid`** (*string*) – Specify the Attachment UUID where *attachments/{uuid}* is the Attachment Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *attachments/add30235-1424-4fda-840a-d5ef82c4c96f*

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to read the attachment.
- **404 Not Found** – Returned when an attachment with the identity does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription’s rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **hash.alg**(*string*) –
- **hash.value**(*string*) – Calculated hash.
- **identity**(*string*) – Relative resource name for the attachment. e.g. attachments/20c97f42-87fc-482c-9d58-4d11abd33359 (read only)
- **mime_type**(*string*) – Type of data e.g. image/jpeg (read only)
- **size**(*string*) – Size of blob in bytes (read only)
- **timestamp_accepted**(*string*) – Timestamp of creation (read only)

ATTACHMENTS V2 API

9.1 Retrieve Attachment

Note: Attachments in the JitsuIn Archivist system are not first-order objects in their own right: they are properties of other objects such as Asset Records or events. Due to this, Attachments **MUST** be retrieved by providing asset or event identity and unique attachment identity. They cannot be listed, filtered, or searched.

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://symsation.1234-5678.nodes.archivist.jitsuin.io
```

Retrieve a specific Attachment found on an asset assets/c04d5ecf-02e0-4be2-a014-ffbbf0e8ddeb

```
$ curl -v \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/attachments/assets/c04d5ecf-02e0-4be2-a014-ffbbf0e8ddeb/
↪08838336-c357-460d-902a-3aba9528dd22
```

To retrieve a specific Attachment found on an event assets/c04d5ecf-02e0-4be2-a014-ffbbf0e8ddeb/events/de834094-f6c3-4e38-9b37-8c61dea312c9 issue following curl command

```
$ curl -v \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/attachments/assets/c04d5ecf-02e0-4be2-a014-ffbbf0e8ddeb/events/
↪de834094-f6c3-4e38-9b37-8c61dea312c9/08838336-c357-460d-902a-3aba9528dd22
```

The response will be a download of the specified attachment

It's also possible to retrieve information about specific attachment using this API. To do that simply issue request as above with a suffix /info

```
$ curl -v \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/attachments/assets/c04d5ecf-02e0-4be2-a014-ffbbf0e8ddeb/
↪08838336-c357-460d-902a-3aba9528dd22/info
```

(continues on next page)

(continued from previous page)

The response will include basic information about the attachment

```
{
  "identity": "attachments/08838336-c357-460d-902a-3aba9528dd22",
  "hash": {
    "alg": "SHA256",
    "value": "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  },
  "mime_type": "image/jpeg",
  "timestamp_accepted": "2019-11-07T15:31:49Z",
  "size": 31424
}
```

BLOBSV1 API

10.1 Upload Blob

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Upload the blob stored at /path/to/file:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "content_type=image/jpeg" \
  -F "file=@/path/to/file" \
  $URL/archivist/v1/blobs
```

The response is:

```
{
  "identity": "blobs/08838336-c357-460d-902a-3aba9528dd22",
  "hash": {
    "alg": "SHA256",
    "value": "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  },
  "mime_type": "image/jpeg",
  "timestamp_accepted": "2019-11-07T15:31:49Z",
  "size": 31424
}
```

Note:

identity The unique identity of the asset in the Jitsuin Archivist system, used to reference the blob in an Asset Record or asset event (usually a Maintenance event).

Warning: Blobs in the Jitsuin Archivist system are not first-order objects in their own right: they are properties of other objects such as Asset Records or events.

10.2 Retrieve Blob

Note: Blobs in the Jitsuin Archivist system are not first-order objects in their own right: they are properties of other objects such as Asset Records or events.

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://symsation.1234-5678.nodes.archivist.jitsuin.io
```

Retrieve a specific Attachment

```
$ curl -v \
  -H "@$BEARER_TOKEN_FILE" \
  -H "content_type=image/jpeg" \
  -F "file=@/path/to/file" \
  $URL/archivist/v1/blobs/08838336-c357-460d-902a-3aba9528dd22
```

The response is:

```
{
  "identity": "blobsV1/08838336-c357-460d-902a-3aba9528dd22",
  "hash": {
    "alg": "SHA256",
    "value": "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  },
  "mime_type": "image/jpeg",
  "timestamp_accepted": "2019-11-07T15:31:49Z",
  "size": 31424
}
```

LOCATIONS API

11.1 Location Creation

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Define the location parameters and store in /path/to/jsonfile:

```
{
  "display_name": "Macclesfield, Cheshire",
  "description": "Manufacturing site, North West England, Macclesfield, Cheshire",
  "latitude": 53.2546799,
  "longitude": -2.1213956,
  "attributes": {
    "director": "John Smith",
    "address": "Unit 6A, Synsation Park, Maccelsfield",
    "Facility Type": "Manufacture",
    "support_email": "support@macclesfield.com",
    "support_phone": "123 456 789"
  }
}
```

Note:

display_name required Friendly name for the location. Displayed in the Archivist GUI.

description required Extended information about the location.

extended_attributes freeform and can contain any fields.

See [Swagger POST API](#)

Create the location to POSTing to the `locations` resource:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/v2/locations
```

The response is:

```
{
  "identity": "locations/08838336-c357-460d-902a-3aba9528dd22",
  "display_name": "Macclesfield, Cheshire",
  "description": "Manufacturing site, North West England, Macclesfield, Cheshire",
  "latitude": 53.2546799,
  "longitude": -2.1213956,
  "attributes": {
    "director": "John Smith",
    "address": "Bridgewater, Somerset",
    "Facility Type": "Manufacture",
    "support_email": "support@macclesfield.com",
    "support_phone": "123 456 789"
  }
}
```

Note:

identity used to attach the location to an asset during asset creation or asset event. (usually a maintenance event).

11.2 Location Retrieval

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

11.2.1 Fetch all locations

To fetch all locations, simply GET the `locations` resource:

```
$ curl -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/locations
```


11.2.2 Fetch specific location by identity

If you know the unique identity of the location record, simply GET the resource:

```
$ curl -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/locations/08838336-c357-460d-902a-3aba9528dd22
```

11.2.3 Fetch location by name

To fetch all locations with a specific name, GET the `assets` resource and filter on `display_name`:

```
$ curl -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/locations?display_name=Macclesfield%2C%20Cheshire
```

Note: It is advisable to implement a unique naming scheme for location names, but this is not enforced by the system. If multiple locations exist with the same name they will all be returned, and the client will need to differentiate based on other attributes.

Each of these calls returns a list of matching asset records in the form:

```
{
  "locations": [
    {
      "identity": "locations/08838336-c357-460d-902a-3aba9528dd22",
      "display_name": "Macclesfield, Cheshire",
      "description": "Manufacturing site, North West England, Macclesfield, ↵
↵Cheshire",
      "latitude": "53.2546799",
      "longitude": "-2.1213956,14.54",
      "attributes": {
        "director": "John Smith",
        "address": "Bridgewater, Somerset",
        "Facility Type": "Manufacture",
        "support_email": "support@macclesfield.com",
        "support_phone": "123 456 789"
      }
    }
  ]
}
```

Note: The number of records returned has a maximum limit. If this limit is too small then one must use [API Request Paging](#).

Note: The total number of assets that exist is returned in the response header field 'x-total-count' if the 'x-request-total-count' header on the request is set to 'true'. The curl option '-i' will emit this to stdout.

Note: A full API reference is available in [Swagger GET API](#)

11.3 Locations Swagger API

GET /archivist/v2/locations

List locations

Returns a paginated list of locations

Query Parameters

- **order_by** (*string*) – Specify the sort order for the results. By display_name and by named extended attribute are supported.
- **page_size** (*integer*) – Maximum locations per page
- **page_token** (*string*) – The next_page_token returned from a previous list request if any.
- **display_name** (*string*) – Further fields are bound to query parameters and act to filter the result
- **description** (*string*) –

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to read the location.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **locations[] .attributes** (*object*) – key value mapping of asset properties
- **locations[] .description** (*string*) – Free text description of the location
- **locations[] .display_name** (*string*) – Customer friendly name for the location.
- **locations[] .identity** (*string*) – Unique identification for the location, Relative Resource Name
- **locations[] .latitude** (*number*) – Latitude in decimal degrees
- **locations[] .longitude** (*number*) – Longitude in decimal degrees
- **locations[] .owner** (*string*) – wallet address of the location owner
- **locations[] .tenant** (*string*) – Tenant id
- **next_page_token** (*string*) – Token to retrieve the next page of results or empty if there are none.

POST /archivist/v2/locations

Create a location

Create a new location.

Request JSON Object

- **attributes** (*object*) – key value mapping of location properties
- **description** (*string*) – Free text description of the location

- **display_name** (*string*) – Customer friendly name for the location. (required)
- **latitude** (*number*) – Latitude in decimal degrees format. (required)
- **longitude** (*number*) – longitude in decimal degrees format. (required)

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **402 Payment Required** – Returned when the user's quota of locations policies has been reached.
- **403 Forbidden** – Returned when the user is not authorized to create a location.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **attributes** (*object*) – key value mapping of asset properties
- **description** (*string*) – Free text description of the location
- **display_name** (*string*) – Customer friendly name for the location.
- **identity** (*string*) – Unique identification for the location, Relative Resource Name
- **latitude** (*number*) – Latitude in decimal degrees
- **longitude** (*number*) – Longitude in decimal degrees
- **owner** (*string*) – wallet address of the location owner
- **tenant** (*string*) – Tenant id

GET /archivist/v2/locations/{uuid}

Get a location

Returns the identified location

Parameters

- **uuid** (*string*) – Specify the Location UUID where *locations/{uuid}* is the Location Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *locations/add30235-1424-4fda-840a-d5ef82c4c96f*

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to read the location.
- **404 Not Found** – Returned when the identified location does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **attributes** (*object*) – key value mapping of asset properties

- **description** (*string*) – Free text description of the location
- **display_name** (*string*) – Customer friendly name for the location.
- **identity** (*string*) – Unique identification for the location, Relative Resource Name
- **latitude** (*number*) – Latitude in decimal degrees
- **longitude** (*number*) – Longitude in decimal degrees
- **owner** (*string*) – wallet address of the location owner
- **tenant** (*string*) – Tenant id

DELETE /archivist/v2/locations/{uuid}**Delete a location**

Delete the identified location

Parameters

- **uuid** (*string*) – Specify the Location UUID where *locations/{uuid}* is the Location Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *locations/add30235-1424-4fda-840a-d5ef82c4c96f*

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to delete the location.
- **404 Not Found** – Returned when the identified location does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

PATCH /archivist/v2/locations/{uuid}**Update a location's details**

Perform a full or partial update of the identified location

Parameters

- **uuid** (*string*) – Specify the Location UUID where *locations/{uuid}* is the Location Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *locations/add30235-1424-4fda-840a-d5ef82c4c96f*

Query Parameters

- **mask** (*string*) – The mask says exactly which fields are to be updated. This removes the ambiguity caused by 'zero' valued fields. <https://grpc-ecosystem.github.io/grpc-gateway/docs/patch.html>

Request JSON Object

- **attributes** (*object*) – key value mapping of asset properties
- **description** (*string*) – Free text description of the location
- **display_name** (*string*) – Customer friendly name for the location.
- **identity** (*string*) – Unique identification for the location, Relative Resource Name
- **latitude** (*number*) – Latitude in decimal degrees

- **longitude** (*number*) – Longitude in decimal degrees
- **owner** (*string*) – wallet address of the location owner
- **tenant** (*string*) – Tenant id

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to update the location.
- **404 Not Found** – Returned when the identified location does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **attributes** (*object*) – key value mapping of asset properties
- **description** (*string*) – Free text description of the location
- **display_name** (*string*) – Customer friendly name for the location.
- **identity** (*string*) – Unique identification for the location, Relative Resource Name
- **latitude** (*number*) – Latitude in decimal degrees
- **longitude** (*number*) – Longitude in decimal degrees
- **owner** (*string*) – wallet address of the location owner
- **tenant** (*string*) – Tenant id

GET /archivist/v2/locations/{uuid}/permissions

Get location permissions

Get location permissions for identified location

Parameters

- **uuid** (*string*) – Specify the Location UUID where *locations/{uuid}* is the Location Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *locations/add30235-1424-4fda-840a-d5ef82c4c96f*

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to list permissions for the location.
- **404 Not Found** – Returned when the identified location does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **location_identity** (*string*) – The location identity in the form: *locations/{uuid}* (read only)

- **permissions.subject_identities[]** (*string*) –

PATCH /archivist/v2/locations/{uuid}/permissions**Patch location permissions**

Patch location permissions for identified location

Parameters

- **uuid** (*string*) – Specify the Location UUID where *locations/{uuid}* is the Location Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *locations/add30235-1424-4fda-840a-d5ef82c4c96f*

Request JSON Object

- **subject_identities[]** (*string*) –

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to access permissions for the location.
- **404 Not Found** – Returned when the identified location does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **location_identity** (*string*) – The location identity in the form: *locations/{uuid}* (read only)
- **permissions.subject_identities[]** (*string*) –

GET /archivist/v2/locations:caps**Get remaining capped resources for Locations**

Not stable or officially supported. Get remaining capped resources for Locations

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **caps[] .resource_remaining** (*string*) – Number of capped resources remaining
- **caps[] .resource_type** (*string*) – String identifying the capped resource type

GET /archivist/v2/locations:openapi**Get OpenAPI spec for Locations**

Get OpenAPI v2.0 spec for Locations

Status Codes

- 200 OK – A successful response.
- 401 Unauthorized – Returned when the user is not authenticated to the system.
- 429 Too Many Requests – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

GET /archivist/v2/locations:openapi-ui
Get OpenAPI UI for Locations

Get OpenAPI v2.0 UI for Locations

Status Codes

- 200 OK – A successful response.
- 401 Unauthorized – Returned when the user is not authenticated to the system.
- 429 Too Many Requests – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

ASSETS API

12.1 Asset Record Creation

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Define the asset parameters and store in /path/to/jsonfile:

```
{
  "behaviours": ["Firmware", "Maintenance", "RecordEvidence", "LocationUpdate",
↪ "Attachments"],
  "attributes": {
    "arc_firmware_version": "1.0",
    "arc_serial_number": "vtl-x4-07",
    "arc_display_name": "tcl.ppj.003",
    "arc_description": "Traffic flow control light at A603 North East",
    "arc_home_location_identity": "locations/115340cf-f39e-4d43-a2ee-8017d672c6c6
↪ ",
    "arc_display_type": "Traffic light with violation camera",
    "some_custom_attribute": "value",
    "arc_attachments": [
      {
        "arc_display_name": "arc_primary_image",
        "arc_attachment_identity": "blobs/87b1a84c-1c6f-442b-923e-a97516f4d275
↪ ",
        "arc_hash_alg": "SHA256",
        "arc_hash_value":
↪ "246c316e2cd6971ce5c83a3e61f9880fa6e2f14ae2976ee03500eb282fd03a60"
      }
    ]
  }
}
```

Note:

behaviours list of behaviours to enable for this asset

attributes properties of asset

See *Behaviours* for details of behaviours and the system- reserved `arc_*` attributes.

See *Upload Blob*, *Location Creation*, and *Location Retrieval* for details of how to get the correct values for `arc_home_location_identity` and `arc_primary_image`.

Create the asset:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/v2/assets
```

The response is:

```
{
  "identity": "assets/3f5be24f-fd1b-40e2-af35-ec7c14c74d53",
  "behaviours": [
    "Firmware",
    "Maintenance",
    "RecordEvidence",
    "LocationUpdate",
    "Attachments"
  ],
  "attributes": {
    "arc_serial_number": "vtl-x4-07",
    "arc_display_name": "tcl.ppj.003",
    "arc_description": "Traffic flow control light at A603 North East",
    "arc_home_location_identity": "locations/115340cf-f39e-4d43-a2ee-8017d672c6c6",
    "arc_display_type": "Traffic light with violation camera",
    "arc_firmware_version": "1.0",
    "some_custom_attribute": "value",
    "arc_attachments": [
      {
        "arc_display_name": "arc_primary_image",
        "arc_attachment_identity": "blobs/87b1a84c-1c6f-442b-923e-a97516f4d275",
        "arc_hash_alg": "SHA256",
        "arc_hash_value": "246c316e2cd6971ce5c83a3e61f9880fa6e2f14ae2976ee03500eb282fd03a60"
      }
    ]
  },
  "confirmation_status": "PENDING",
  "tracked": "TRACKED"
}
```

Note: A full API reference is available in [Swagger POST API](#)

12.2 Asset Record Retrieval

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Asset records in Jitsuin Archivist are tokenized at creation time and referred to in all API calls and smart contracts throughout the system by a unique identity of the form:

```
assets/12345678-90ab-cdef-1234-567890abcdef.
```

If you do not know the asset's identity you can fetch asset records using other information you do know, such as the asset's name in your asset management or digital twin platform.

12.2.1 Fetch all assets

To fetch all asset records, simply GET the `assets` resource:

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets
```

12.2.2 Fetch specific asset by identity

If you know the unique identity of the Asset Record simply GET the resource:

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets/6a951b62-0a26-4c22-a886-1082297b063b
```

12.2.3 Fetch specific asset at given point in time by identity

If you know the unique identity of an Asset Record and want to show its state at any given point in the past, simply GET with following query parameter

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets/6a951b62-0a26-4c22-a886-1082297b063b?at_time=2021-01-
  13T12:34:21Z
```

This will return the Asset Record with the values it had on 2021-01-13T12:34:21Z

12.2.4 Fetch assets by name

To fetch all assets with a specific name, GET the assets resource and filter on `arc_display_name`:

```
$ curl -g -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets?attributes.arc_display_name=tcl.ccj.003
```

12.2.5 Fetch assets by type

To fetch all assets of a specific type, GET the assets resource and filter on `arc_display_type`:

```
$ curl -g -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets?attributes.arc_display_type=Traffic%20light
```

Each of these calls returns a list of matching asset records in the form:

```
{
  "assets": [
    {
      "identity": "assets/6a951b62-0a26-4c22-a886-1082297b063b",
      "behaviours": [
        "Firmware",
        "Maintenance",
        "RecordEvidence",
        "LocationUpdate",
        "Attachments"
      ],
      "attributes": {
        "arc_display_type": "Traffic light",
        "arc_firmware_version": "1.0",
        "arc_home_location_identity": "locations/866790d8-4ed6-4cc9-8f60-
↪07672609b331",
        "arc_serial_number": "vtl-x4-07",
        "arc_description": "Traffic flow control light at A603 North East",
        "arc_display_name": "tcl.ccj.003",
        "some_custom_attribute": "value",
        "arc_attachments": [
          {
            "arc_display_name": "arc_primary_image",
            "arc_attachment_identity": "blobs/87b1a84c-1c6f-442b-923e-
↪a97516f4d275",
            "arc_hash_alg": "SHA256",
            "arc_hash_value":
↪"246c316e2cd6971ce5c83a3e61f9880fa6e2f14ae2976ee03500eb282fd03a60"
          }
        ]
      },
      "confirmation_status": "CONFIRMED",
      "tracked": "TRACKED"
    }
  ]
}
```

12.2.6 Fetch assets by filtering for presence of a field

To fetch all assets with a field set to any value, GET the `assets` resource and filter on most available fields. For example:

```
$ curl -g -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets?attributes.arc_display_name=*
```

Returns all assets which have `arc_display_name` that is not empty.

12.2.7 Fetch assets which are missing a field

To fetch all assets with a field which is not set to any value, GET the `assets` resource and filter on most available fields. For example:

```
$ curl -g -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets?attributes.arc_display_name!=*
```

Returns all assets which do not have `arc_display_name` or in which `arc_display_name` is empty.

Note: See *Behaviours* and *LifeCycle* for details of how to interpret the system-reserved `arc_*` attributes.

See *Upload Blob* and/or *Location Creation* for details of how to handle the `arc_home_location_identity` and `arc_primary_image` attributes.

Note: The number of records returned has a maximum limit. If this limit is too small then one must use *API Request Paging*.

Note: The total number of assets that exist is returned in the response header field 'x-total-count' if the 'x-request-total-count' header on the request is set to 'true'. The curl option '-i' will emit this to stdout.

Note: A full API reference is available in *Swagger GET API*

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Note: A full description of the workflow regarding attaching a file to an asset or event is given in *Attachments*.

Note: The following operations assume that an attachment has been uploaded to Archivist node using the API *Upload Blob*. This attachment uuid is generically referred to as blobs/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx in the following text.

Each attachment has an associated hash value and the name of the hash algorithm used.

12.3 Attachments Operations API

12.3.1 Attachments Attach

Define the event parameters and store in /path/to/jsonfile:

```
{
  "operation": "Attach",
  "behaviour": "Attachments",
  "event_attributes": {
    "arc_append_attachments": [
      {
        "arc_attachment_identity": "blobs/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx",
        "arc_display_name": "an attachment 1",
        "arc_hash_value": "jnwpldckndwwlskqaalijopjkkkkoji1",
        "arc_hash_alg": "sha256",
      },
      {
        "arc_attachment_identity": "blobs/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx",
        "arc_display_name": "an attachment 2",
        "arc_hash_value": "042aea10a0f14f2d391373599be69d53a75dde9951fc3d3cd10b6100aa7a9f24",
        "arc_hash_alg": "sha256",
      }
    ]
  },
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "principal_declared": {
    "issuer": "idp.synsation.io/1234",
    "subject": "phil.b",
    "email": "phil.b@synsation.io"
  }
}
```

Note:

event_attributes.arc_append_attachments *Required* List with details of all attachments to be attached to an asset, each attachment details must have following four fields defined:

arc_attachment_identity *Required* identity of an attachment

arc_display_name *Required* display name of an attachment

arc_hash_value *Required* hash of the attachment Content

arc_hash_alg *Required* algorithm used to calculate the hash

timestamp_declared *Optional* Client-claimed time at which the maintenance was performed

principal_declared *Optional* Client-claimed identity of person performing the operation

Add the Attachments request to the Asset Record by POSTing it to the resource:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/v2/assets/add30235-1424-4fda-840a-d5ef82c4c96f/events
```

The response is:

```
{
  "identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-
  ↪8dcf-dc8c4aefc000",
  "asset_identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f",
  "operation": "Attach",
  "behaviour": "Attachments",
  "event_attributes": {
    "arc_append_attachments": [
      {
        "arc_attachment_identity": "blobs/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx",
        "arc_display_name": "an attachment 1",
        "arc_hash_value": "jnwppjocoqsssnundwlqalsqiiqsqp;
        ↪lpwpldkndwwlskqaalijopjkkkkkojijl",
        "arc_hash_alg": "sha256",
      },
      {
        "arc_attachment_identity": "blobs/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx",
        "arc_display_name": "an attachment 2",
        "arc_hash_value":
        ↪"042aea10a0f14f2d391373599be69d53a75dde9951fc3d3cd10b6100aa7a9f24",
        "arc_hash_alg": "sha256",
      }
    ],
  },
  "asset_attributes": {
    "arc_attachments": [
      {
        "arc_attachment_identity": "blobs/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx",
        "arc_display_name": "an attachment 1",
        "arc_hash_value": "jnwppjocoqsssnundwlqalsqiiqsqp;
        ↪lpwpldkndwwlskqaalijopjkkkkkojijl",
        "arc_hash_alg": "sha256",
      },
      {
        "arc_attachment_identity": "blobs/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx",
        "arc_display_name": "an attachment 2",
        "arc_hash_value":
        ↪"042aea10a0f14f2d391373599be69d53a75dde9951fc3d3cd10b6100aa7a9f24",
        "arc_hash_alg": "sha256",
      }
    ],
  },
  "timestamp_accepted": "2019-11-27T15:13:21Z",
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "timestamp_committed": "2019-11-27T15:15:02Z",
}
```

(continues on next page)

(continued from previous page)

```
"principal_declared": {
  "issuer": "idp.synsation.io/1234",
  "subject": "phil.b",
  "email": "phil.b@synsation.io"
},
"principal_accepted": {
  "issuer": "job.idp.server/1234",
  "subject": "bob@job"
},
"confirmation_status": "CONFIRMED",
"block_number": 12,
"transaction_index": 5,
"transaction_id": "0x07569"
}
```

Note:

asset_attributes.arc_attachments Holds all asset attachments - pre-existing asset attachments and attachments provided in `arc_append_attachments`

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *[API Request Authorization and Authentication](#)*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

12.4 Builtin Operations API

12.4.1 Builtin Add

Define the event parameters and store in `/path/to/jsonfile`:

```
{
  "operation": "Add",
  "behaviour": "Builtin",
  "event_attributes": {
    "arc_behaviour_name": "Attachments"
  },
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "principal_declared": {
    "issuer": "idp.synsation.io/1234",
    "subject": "phil.b",
    "email": "phil.b@synsation.io"
  }
}
```


Note:

event_attributes.arc_behaviour_name *Required* Name of the behaviour to be added to asset

timestamp_declared *Optional* Client-claimed time at which the maintenance was performed

principal_declared *Optional* Client-claimed identity of person performing the operation

Add the Builtin request to the Asset Record by POSTing it to the resource:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/v2/assets/add30235-1424-4fda-840a-d5ef82c4c96f/events
```

The response is:

```
{
  "identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-
↪8dcf-dc8c4aefc000",
  "asset_identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f",
  "operation": "Add",
  "behaviour": "Builtin",
  "event_attributes": {
    "arc_behaviour_name": "Attachments"
  },
  "timestamp_accepted": "2019-11-27T15:13:21Z",
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "timestamp_committed": "2019-11-27T15:15:02Z",
  "principal_declared": {
    "issuer": "idp.synsation.io/1234",
    "subject": "phil.b",
    "email": "phil.b@synsation.io"
  },
  "principal_accepted": {
    "issuer": "job.idp.server/1234",
    "subject": "bob@job"
  },
  "confirmation_status": "CONFIRMED",
  "block_number": 12,
  "transaction_index": 5,
  "transaction_id": "0x07569"
}
```

12.4.2 Builtin Remove

Define the event parameters and store in /path/to/jsonfile:

```
{
  "operation": "Remove",
  "behaviour": "Builtin",
  "event_attributes": {
    "arc_behaviour_name": "Attachments"
  },
  "timestamp_declared": "2019-11-27T14:44:19Z",
```

(continues on next page)

(continued from previous page)

```
"principal_declared": {
  "issuer": "idp.synsation.io/1234",
  "subject": "phil.b",
  "email": "phil.b@synsation.io"
}
```

Note:

event_attributes.arc_behaviour_name *Required* Name of the behaviour to be added to asset

timestamp_declared *Optional* Client-claimed time at which the maintenance was performed

principal_declared *Optional* Client-claimed identity of person performing the operation

Add the Builtin request to the Asset Record by POSTing it to the resource:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/v2/assets/add30235-1424-4fda-840a-d5ef82c4c96f/events
```

The response is:

```
{
  "identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-
↪8dcf-dc8c4aefc000",
  "asset_identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f",
  "operation": "Remove",
  "behaviour": "Builtin",
  "event_attributes": {
    "arc_behaviour_name": "Attachments"
  },
  "timestamp_accepted": "2019-11-27T15:13:21Z",
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "timestamp_committed": "2019-11-27T15:15:02Z",
  "principal_declared": {
    "issuer": "idp.synsation.io/1234",
    "subject": "phil.b",
    "email": "phil.b@synsation.io"
  },
  "principal_accepted": {
    "issuer": "job.idp.server/1234",
    "subject": "bob@job"
  },
  "confirmation_status": "CONFIRMED",
  "block_number": 12,
  "transaction_index": 5,
  "transaction_id": "0x07569"
}
```

12.4.3 Builtin StartTracking

Define the event parameters and store in /path/to/jsonfile:

```
{
  "operation": "StartTracking",
  "behaviour": "Builtin",
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "principal_declared": {
    "issuer": "idp.synsation.io/1234",
    "subject": "phil.b",
    "email": "phil.b@synsation.io"
  }
}
```

Note:

timestamp_declared *Optional* Client-claimed time at which the maintenance was performed

principal_declared *Optional* Client-claimed identity of person performing the operation

Add the Builtin request to the Asset Record by POSTing it to the resource:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/v2/assets/add30235-1424-4fda-840a-d5ef82c4c96f/events
```

The response is:

```
{
  "identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000",
  "asset_identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f",
  "operation": "StartTracking",
  "behaviour": "Builtin",
  "timestamp_accepted": "2019-11-27T15:13:21Z",
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "timestamp_committed": "2019-11-27T15:15:02Z",
  "principal_declared": {
    "issuer": "idp.synsation.io/1234",
    "subject": "phil.b",
    "email": "phil.b@synsation.io"
  },
  "principal_accepted": {
    "issuer": "job.idp.server/1234",
    "subject": "bob@job"
  },
  "confirmation_status": "CONFIRMED",
  "block_number": 12,
  "transaction_index": 5,
  "transaction_id": "0x07569"
}
```

12.4.4 Builtin StopTracking

Define the event parameters and store in /path/to/jsonfile:

```
{
  "operation": "StopTracking",
  "behaviour": "Builtin",
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "principal_declared": {
    "issuer": "idp.synsation.io/1234",
    "subject": "phil.b",
    "email": "phil.b@synsation.io"
  }
}
```

Note:

timestamp_declared *Optional* Client-claimed time at which the maintenance was performed

principal_declared *Optional* Client-claimed identity of person performing the operation

Add the Builtin request to the Asset Record by POSTing it to the resource:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/v2/assets/add30235-1424-4fda-840a-d5ef82c4c96f/events
```

The response is:

```
{
  "identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000",
  "asset_identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f",
  "operation": "StopTracking",
  "behaviour": "Builtin",
  "timestamp_accepted": "2019-11-27T15:13:21Z",
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "timestamp_committed": "2019-11-27T15:15:02Z",
  "principal_declared": {
    "issuer": "idp.synsation.io/1234",
    "subject": "phil.b",
    "email": "phil.b@synsation.io"
  },
  "principal_accepted": {
    "issuer": "job.idp.server/1234",
    "subject": "bob@job"
  },
  "confirmation_status": "CONFIRMED",
  "block_number": 12,
  "transaction_index": 5,
  "transaction_id": "0x07569"
}
```

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

12.5 RecordEvidence Operations API

12.5.1 RecordEvidence Record

Define the event parameters and store in /path/to/jsonfile:

```
{
  "operation": "Record",
  "behaviour": "RecordEvidence",
  "event_attributes": {
    "arc_description": "Safety conformance approved for version 1.6. See attached_
    ↪conformance report",
    "arc_evidence": "DVA Conformance Report attached",
    "conformance_report": "blobs/e2a1d16c-03cd-45a1-8cd0-690831df1273"
  },
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "principal_declared": {
    "issuer": "idp.synsation.io/1234",
    "subject": "phil.b",
    "email": "phil.b@synsation.io"
  }
}
```

Note:

attributes.arc_description *Required* Details of the RecordEvidence request

attributes.arc_evidence *Required* The evidence to be retained in the asset history

attributes.conformance_report *Example* Client can add any additional information in further attributes, including free text or attachments

timestamp_declared *Optional* Client-claimed time at which the maintenance was performed

principal_declared *Optional* Client-claimed identity of person performing the operation

Add the RecordEvidence request to the Asset Record by POSTing it to the resource:

```
$ curl -v -X POST \
  -H "@$BEARER_TOKEN_FILE" \
  -H "Content-type: application/json" \
  -d "@/path/to/jsonfile" \
  $URL/archivist/v2/assets/add30235-1424-4fda-840a-d5ef82c4c96f/events
```

The response is:

```

{
  "identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-
↪8dcf-dc8c4aefc000",
  "asset_identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f",
  "operation": "Record",
  "behaviour": "RecordEvidence",
  "event_attributes": {
    "arc_description": "Safety conformance approved for version 1.6. See attached_
↪conformance report",
    "arc_evidence": "DVA Conformance Report attached",
    "conformance_report": "blobs/e2a1d16c-03cd-45a1-8cd0-690831df1273"
  },
  "timestamp_accepted": "2019-11-27T15:13:21Z",
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "timestamp_committed": "2019-11-27T15:15:02Z",
  "principal_declared": {
    "issuer": "idp.synsation.io/1234",
    "subject": "phil.b",
    "email": "phil.b@synsation.io"
  },
  "principal_accepted": {
    "issuer": "job.idp.server/1234",
    "subject": "bob@job"
  },
  "confirmation_status": "CONFIRMED",
  "block_number": 12,
  "transaction_index": 5,
  "transaction_id": "0x07569"
}

```

12.6 Assets Swagger API

GET /archivist/v2/assets

List Archivist assets

Retrieves a list of Archivist assets

Query Parameters

- **attributes.arc_display_name** (*string*) – List only Assets with this friendly name
- **attributes.arc_display_type** (*string*) – List only Assets of this type
- **page_size** (*integer*) – Maximum results per page.
- **page_token** (*string*) – The next_page_token returned from a previous list request if any.
- **order_by** (*string*) – Specify the sort order for the results.
- **tracked** (*string*) – indicates whether asset is still being tracked in the system
- **confirmation_status** (*string*) – indicates if the asset has been successfully committed to the blockchain
- **storage_integrity** (*string*) – XXX: #4483 DEPRECATED use proof_mechanism DEPRECATED use proof_mechanism

- **proof_mechanism** (*string*) – proof mechanism of the asset (and all its events)
the mechanism used to provide evidential proof
- **chain_id** (*string*) – chain id of the blockchain associated with this asset
- **privacy** (*string*) – privacy filter of the asset (and all its events)
the privacy status of the asset

Status Codes

- **200 OK** – A successful response.
- **206 Partial Content** – The number of assets exceeds the servers limit. The approximate number of matching results is provided by the x-total-count header if the 'x-request-total-count' header on the request is set to 'true'. The exact limit is available in the content-range header. The value format is 'items 0-LIMIT/TOTAL'. Note that x-total-count is always present for 200 and 206 responses. It is the servers best available approximation. Similarly, in any result set, you may get a few more than LIMIT items.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to list Assets.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **assets[]** .**at_time** (*string*) – indicates time the asset data is from (read only)
- **assets[]** .**attributes** (*object*) – key value mapping of asset properties
- **assets[]** .**behaviours[]** (*string*) –
- **assets[]** .**chain_id** (*string*) – chain id of the blockchain associated with this asset (read only)
- **assets[]** .**confirmation_status** (*string*) –
- **assets[]** .**identity** (*string*) – relative resource address *assets/{UUID}* (read only)
- **assets[]** .**owner** (*string*) – wallet address of the asset owner (read only)
- **assets[]** .**proof_mechanism** (*string*) –
- **assets[]** .**public** (*boolean*) – Public asset
- **assets[]** .**storage_integrity** (*string*) – Specifies how the asset data will be stored. This is set once on creation and does not change.
- **assets[]** .**tenant_identity** (*string*) – Identity of the tenant the that created this asset
- **assets[]** .**tracked** (*string*) –
- **next_page_token** (*string*) – Token to retrieve the next page of results or empty if there are none.

POST /archivist/v2/assets

Create an Archivist asset

Creates an Archivist asset

Request JSON Object

- **attributes** (*object*) – key value mapping of asset attributes
- **behaviours** [*string*] –
- **public** (*boolean*) – Public asset. A public asset and all its events are visible to the general public. Sharing to specific organisations is not available for public assets.

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **402 Payment Required** – Returned when the user either has not enabled blockchain storage or the number of assets would exceed the user's quota
- **403 Forbidden** – Returned when the user is not authorized to create an Asset.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **at_time** (*string*) – indicates time the asset data is from (read only)
- **attributes** (*object*) – key value mapping of asset properties
- **behaviours** [*string*] –
- **chain_id** (*string*) – chain id of the blockchain associated with this asset (read only)
- **confirmation_status** (*string*) –
- **identity** (*string*) – relative resource address *assets/{UUID}* (read only)
- **owner** (*string*) – wallet address of the asset owner (read only)
- **proof_mechanism** (*string*) –
- **public** (*boolean*) – Public asset
- **storage_integrity** (*string*) – Specifies how the asset data will be stored. This is set once on creation and does not change.
- **tenant_identity** (*string*) – Identity of the tenant that created this asset
- **tracked** (*string*) –

GET /archivist/v2/assets/{uuid}

Retrieves a specific Archivist asset

Retrieves a specific Archivist asset

Parameters

- **uuid** (*string*) – Specify the Asset UUID where *assets/{uuid}* is the Asset Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *assets/add30235-1424-4fda-840a-d5ef82c4c96f*

Query Parameters

- **at_time** (*string*) – Specify time in the past to show asset data as it was at time specified

Status Codes

- **200 OK** – A successful response.

- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to view an Asset.
- **404 Not Found** – Returned when the asset with the id does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **at_time** (*string*) – indicates time the asset data is from (read only)
- **attributes** (*object*) – key value mapping of asset properties
- **behaviours[]** (*string*) –
- **chain_id** (*string*) – chain id of the blockchain associated with this asset (read only)
- **confirmation_status** (*string*) –
- **identity** (*string*) – relative resource address *assets/{UUID}* (read only)
- **owner** (*string*) – wallet address of the asset owner (read only)
- **proof_mechanism** (*string*) –
- **public** (*boolean*) – Public asset
- **storage_integrity** (*string*) – Specifies how the asset data will be stored. This is set once on creation and does not change.
- **tenant_identity** (*string*) – Identity of the tenant the that created this asset
- **tracked** (*string*) –

EVENTS API

13.1 Event Record Creation

Events are created on an asset via the behaviour interfaces such as `RecordEvidence`, `Firmware` or `Maintenance`.

See *Assets API*.

13.2 Event Record Retrieval

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Event records in Jitsuin Archivist are tokenized at creation time and referred to in all API calls and smart contracts throughout the system by a unique identity of the form:

```
assets/12345678-90ab-cdef-1234-567890abcdef/events/11bf5b37-e0b8-42e0-8dcf-  
↪dc8c4aefc000
```

If you do not know the event's identity you can fetch event records using other information you do know, such as the event's name in your asset management or digital twin platform.

13.2.1 Fetch all events

To fetch all event records, simply GET the `events` resource:

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets/-/events
```

13.2.2 Fetch events by asset identity

If you know the unique identity of the Asset Record simply GET the resource:

```
$ curl -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets/6a951b62-0a26-4c22-a886-1082297b063b/events
```

13.2.3 Fetch events by behaviour name

To fetch all events with a specific behaviour, GET the `events` resource and filter on behaviour:

```
$ curl -g -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets/-/events?behaviour=firmware
```

13.2.4 Fetch events by behaviour name and operation

To fetch all events with a specific behaviour and operation, GET the `events` resource and filter on firmware and update:

```
$ curl -g -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets/-/events?behaviour=firmware&operation=update
```

13.2.5 Fetch events by filtering on field names

To fetch all events with a specific characteristics, GET the `events` resource and filter on most available fields. For example:

```
$ curl -g -v -X GET \
  -H "$BEARER_TOKEN_FILE" \
  $URL/archivist/v2/assets/-/events?attributes.arc_firmware_version=1.23
```

13.2.6 Fetch events by filtering on dates

To fetch all events occurring at different dates, GET the `events` resource and filter on timestamps. For example:

```
$ curl -g -v -X GET \
  -H "$@${BEARER_TOKEN_FILE}" \
  $URL/archivist/v2/assets/-/events?timestamp_accepted_before=2019-12-01T00:00:00Z
```

Other timestamp filters are `timestamp_accepted_since`, `timestamp_committed_before`, `timestamp_committed_since`, `timestamp_declared_before`, `timestamp_declared_since`.

13.2.7 Responses

Each of these calls returns a list of matching event records in the form:

```
{
  "identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000",
  "asset_identity": "assets/add30235-1424-4fda-840a-d5ef82c4c96f",
  "behaviour": "Firmware",
  "operation": "Update",
  "event_attributes": {
    "arc_description": "Patched during regular patch window",
    "arc_correlation_value": "12-345-67"
  },
  "asset_attributes": {
    "arc_firmware_version": "3.2.1",
  },
  "timestamp_accepted": "2019-11-27T15:13:21Z",
  "timestamp_declared": "2019-11-27T14:44:19Z",
  "timestamp_committed": "2019-11-27T15:15:02Z",
  "principal_declared": {
    "issuer": "idp.synsation.io/1234",
    "subject": "phil.b",
    "email": "phil.b@synsation.io"
  },
  "principal_accepted": {
    "issuer": "job.idp.server/1234",
    "subject": "bob@job"
  },
  "confirmation_status": "CONFIRMED",
  "block_number": 12,
  "transaction_index": 5,
  "transaction_id": "0x07569"
}
```

Note: See *Behaviours* and *LifeCycle* for details of how to interpret the system-reserved `arc_*` attributes.

Note: The number of records returned has a maximum limit. If this limit is too small then one must use *API Request Paging*.

Note: The total number of assets that exist is returned in the response header field 'x-total-count' if the 'x-request-total-count' header on the request is set to 'true'. The curl option '-i' will emit this to stdout.

Note: A full API reference is available in [Swagger GET API](#)

13.3 Events Swagger API

GET /archivist/v2/assets/{uuid}/events

List Archivist events

Lists Archivist events

Parameters

- **uuid** (*string*) – Specify the Asset UUID where *assets/{uuid}* is the Asset Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *assets/add30235-1424-4fda-840a-d5ef82c4c96f*. Use - to list Events for all Assets

Query Parameters

- **attributes.arc_display_type** (*string*) – Only list Events matching this event type
- **page_size** (*integer*) – Maximum results per page.
- **page_token** (*string*) – The next_page_token returned from a previous list request if any.
- **order_by** (*string*) – Specify the sort order for the results.
- **confirmation_status** (*string*) – indicates if the asset has been successfully committed to the blockchain
- **principal_declared.issuer** (*string*) – optional issuer of the principal identity. Where the issuer is not provided the subject is treated as a free string
- **principal_declared.subject** (*string*) – unique identifier of the principal (within issuer context)
- **principal_declared.display_name** (*string*) – The displayable name of the end-user. The name claim is preferred, followed by email claims, then a composite of given_name, middle_name, family_name
- **principal_declared.email** (*string*) – The email for the end-user if available. If email_verified is available it is preferred. Empty if neither email_verified or email are provided by the IdP
- **principal_accepted.issuer** (*string*) – optional issuer of the principal identity. Where the issuer is not provided the subject is treated as a free string
- **principal_accepted.subject** (*string*) – unique identifier of the principal (within issuer context)
- **principal_accepted.display_name** (*string*) – The displayable name of the end-user. The name claim is preferred, followed by email claims, then a composite of given_name, middle_name, family_name

- **principal_accepted.email** (*string*) – The email for the end-user if available. If email_verified is available it is preferred. Empty if neither email_verified or email are provided by the IdP
- **timestamp_accepted_since** (*string*) – Filtering
Only list events when timestamp_accepted is greater
- **timestamp_accepted_before** (*string*) – Only list events when timestamp_accepted is lesser
- **timestamp_committed_since** (*string*) – Only list events when timestamp_committed is greater
- **timestamp_committed_before** (*string*) – Only list events when timestamp_committed is lesser
- **timestamp_declared_since** (*string*) – Only list events when timestamp_declared is greater
- **timestamp_declared_before** (*string*) – Only list events when timestamp_declared is lesser
- **operation** (*string*) –
- **behaviour** (*string*) –
- **proof_mechanism** (*string*) – mechanism for evidential proof for Events on this Asset
specify the mechanism used to provide evidential proof for Events on this Asset

Status Codes

- **200 OK** – A successful response.
- **206 Partial Content** – The number of events exceeds the servers limit. The approximate number of matching results is provided by the x-total-count header, the exact limit is available in the content-range header. The value format is 'items 0-LIMIT/TOTAL'. Note that x-total-count is always present for 200 and 206 responses. It is the servers best available approximation. Similarly, in any result set, you may get a few more than LIMIT items.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to list Events.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **events[] .asset_attributes** (*object*) – key value mapping of asset attributes
- **events[] .asset_identity** (*string*) – identity of a related asset resource
assets/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000 (read only)
- **events[] .behaviour** (*string*) – The behaviour used to create event. *RecordEvidence* (read only)
- **events[] .block_number** (*string*) – number of block event was committed on (read only)
- **events[] .confirmation_status** (*string*) –
- **events[] .event_attributes** (*object*) – key value mapping of event attributes

- **events[] .from** (*string*) – wallet address for the creator of this event (read only)
- **events[] .identity** (*string*) – identity of a event resource (read only)
- **events[] .operation** (*string*) – The operation represented by the event. *Record* (read only)
- **events[] .principal_accepted.display_name** (*string*) – The displayable name of the end-user. The name claim is preferred, followed by email claims, then a composite of given_name, middle_name, family_name
- **events[] .principal_accepted.email** (*string*) – The email for the end-user if available. If email_verified is available it is preferred. Empty if neither email_verified or email are provided by the IdP
- **events[] .principal_accepted.issuer** (*string*) – optional issuer of the principal identity. Where the issuer is not provided the subject is treated as a free string
- **events[] .principal_accepted.subject** (*string*) – unique identifier of the principal (within issuer context)
- **events[] .principal_declared.display_name** (*string*) – The displayable name of the end-user. The name claim is preferred, followed by email claims, then a composite of given_name, middle_name, family_name
- **events[] .principal_declared.email** (*string*) – The email for the end-user if available. If email_verified is available it is preferred. Empty if neither email_verified or email are provided by the IdP
- **events[] .principal_declared.issuer** (*string*) – optional issuer of the principal identity. Where the issuer is not provided the subject is treated as a free string
- **events[] .principal_declared.subject** (*string*) – unique identifier of the principal (within issuer context)
- **events[] .tenant_identity** (*string*) – Identity of the tenant the that created this event
- **events[] .timestamp_accepted** (*string*) – time of event as recorded by the server (read only)
- **events[] .timestamp_committed** (*string*) – time of event as recorded on blockchain (read only)
- **events[] .timestamp_declared** (*string*) – time of event as declared by the user (read only)
- **events[] .transaction_id** (*string*) – hash of the transaction as a hex string *0x11bf5b37e0b842e08dcfdc8c4aefc000*
- **events[] .transaction_index** (*string*) – index of event within committed block (read only)
- **next_page_token** (*string*) – Token to retrieve the next page of results or empty if there are none.

GET /archivist/v2/assets/{asset_uuid}/events/{uuid}

Retrieves Archivist event

Retrieves a specific Archivist event

Parameters

- **asset_uuid** (*string*) – Specify the Asset UUID where *assets/{asset_uuid}/events/{uuid}* is the Event Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000*
- **uuid** (*string*) – Specify the Event UUID where *assets/{asset_uuid}/events/{uuid}* is the Event Identity e.g. *11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000* from Identity *assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000*

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to view Event.
- **404 Not Found** – Returned when the event does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **asset_attributes** (*object*) – key value mapping of asset attributes
- **asset_identity** (*string*) – identity of a related asset resource *assets/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000* (read only)
- **behaviour** (*string*) – The behaviour used to create event. *RecordEvidence* (read only)
- **block_number** (*string*) – number of block event was committed on (read only)
- **confirmation_status** (*string*) –
- **event_attributes** (*object*) – key value mapping of event attributes
- **from** (*string*) – wallet address for the creator of this event (read only)
- **identity** (*string*) – identity of a event resource (read only)
- **operation** (*string*) – The operation represented by the event. *Record* (read only)
- **principal_accepted.display_name** (*string*) – The displayable name of the end-user. The name claim is preferred, followed by email claims, then a composite of given_name, middle_name, family_name
- **principal_accepted.email** (*string*) – The email for the end-user if available. If email_verified is available it is preferred. Empty if neither email_verified or email are provided by the IdP
- **principal_accepted.issuer** (*string*) – optional issuer of the principal identity. Where the issuer is not provided the subject is treated as a free string
- **principal_accepted.subject** (*string*) – unique identifier of the principal (within issuer context)
- **principal_declared.display_name** (*string*) – The displayable name of the end-user. The name claim is preferred, followed by email claims, then a composite of given_name, middle_name, family_name
- **principal_declared.email** (*string*) – The email for the end-user if available. If email_verified is available it is preferred. Empty if neither email_verified or email are provided by the IdP

- **principal_declared.issuer** (*string*) – optional issuer of the principal identity. Where the issuer is not provided the subject is treated as a free string
- **principal_declared.subject** (*string*) – unique identifier of the principal (within issuer context)
- **tenant_identity** (*string*) – Identity of the tenant the that created this event
- **timestamp_accepted** (*string*) – time of event as recorded by the server (read only)
- **timestamp_committed** (*string*) – time of event as recorded on blockchain (read only)
- **timestamp_declared** (*string*) – time of event as declared by the user (read only)
- **transaction_id** (*string*) – hash of the transaction as a hex string *0x11bf5b37e0b842e08dcfdc8c4aefc000*
- **transaction_index** (*string*) – index of event within committed block (read only)

GET /archivist/v2/assets/{asset_uuid}/events/{uuid}:publicurl
Retrieves the public url for a specific Archivist event.

Retrieves the public url for a specific Archivist event.

Parameters

- **asset_uuid** (*string*) – Specify the Asset UUID where *assets/{asset_uuid}/events/{uuid}* is the Event Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000*
- **uuid** (*string*) – Specify the Event UUID where *assets/{asset_uuid}/events/{uuid}* is the Event Identity e.g. *11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000* from Identity *assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000*

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to view an Asset.
- **404 Not Found** – Returned when the asset with the id does not exist.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **publicurl** (*string*) –

BLOCKCHAIN API (V1ALPHA2)

14.1 Blockchain Retrieval (v1alpha2)

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://synsation.1234-5678.nodes.archivist.jitsuin.io
```

Blockchain transactions in Jitsuin Archivist are available via this interface. The transactions are available from the **blockchain** endpoint using the assets event Id as a parameter:

```
assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-8dcf-  
↳dc8c4aefc000
```

14.1.1 Fetch Transactions for an event (v1alpha2)

To fetch all transactions for an asset event GET the **blockchain** resource:

```
$ curl -v -X GET \  
  -H "@$BEARER_TOKEN_FILE" \  
  $URL/archivist/v1alpha2/blockchain/assets/add30235-1424-4fda-840a-d5ef82c4c96f/  
↳events/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000
```

Each of these calls returns a list of matching blockchain transactions in the form:

```
{  
  "transactions": [  
    {  
      "hash":  
↳"0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b",  
      "nonce": 2,  
      "blockHash":  
↳"0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",  
      "blockNumber": 3,  
      "transactionIndex": 0,  
    }  
  ]  
}
```

(continues on next page)

(continued from previous page)

```

        "r": "0x8912348621879462817634897216348712638941",
        "s": "0x1234689712638957682375682364892376487238",
        "from": "0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b",
        "to": "0x6295ee1b4f6dd65047762f924ecd367c17eabf8f",
        "value": "123450000000000000",
        "gas": 314159,
        "gasPrice": "2000000000000",
        "input": "0x57cb2fc4",
        "v": "0x26"
    },
    {
        "hash":
        ↪ "0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b",
        "nonce": 2,
        "blockHash":
        ↪ "0xef1234567d3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
        "blockNumber": 3,
        "transactionIndex": 0,
        "r": "0x8912348621879462817634897216348712638941",
        "s": "0x1234689712638957682375682364892376487238",
        "from": "0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b",
        "to": "0x6295ee1b4f6dd65047762f924ecd367c17eabf8f",
        "value": "123450000000000000",
        "gas": 314159,
        "gasPrice": "2000000000000",
        "input": "0x57cb2fc4",
        "v": "0x26"
    }
]
}

```

Note: The number of records returned has a maximum limit. If this limit is too small then one must use [API Request Paging](#).

A full API reference is available in [Swagger GET API](#)

14.2 Blockchain Swagger API

GET /archivist/v1alpha2/blockchain/assets/{asset_uuid}/events/{uuid}

List of Blockchain Transactions associated with an event.

List of Blockchain Transactions associated with an event. If the event's asset has a proof mechanism of khipu, this will be a list of transactions that compose the event. If the event's asset has a proof mechanism of simple hash, this will be a list containing one transaction, that describes the simple hash anchor transaction. If the list is empty, the event has not been anchored yet.

Parameters

- **asset_uuid** (*string*) – Specify the Asset UUID where *assets/{asset_uuid}/events/{uuid}* is the Event Identity e.g. *add30235-1424-4fda-840a-d5ef82c4c96f* from Identity *assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000*

- **uuid** (*string*) – Specify the Event UUID where *assets/{asset_uuid}/events/{uuid}* is the Event Identity e.g. *11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000* from Identity *assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-8dcf-dc8c4aefc000*

Query Parameters

- **page_size** (*integer*) – Maximum entries per page
- **page_token** (*string*) – The next_page_token returned from a previous list request if any.

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to view event's blockchain transactions.
- **404 Not Found** – Returned when the asset with the id does not exist. or the event with the id does not exist
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **next_page_token** (*string*) – Token to retrieve the next page of results or empty if there are none.
- **transactions[]**.**kipu_details** (*object*) –
- **transactions[]**.**kind** (*string*) –
- **transactions[]**.**simple_hash_details.anchor_hash** (*string*) –
- **transactions[]**.**simple_hash_details.api_query** (*string*) –
- **transactions[]**.**simple_hash_details.end_time** (*string*) –
- **transactions[]**.**simple_hash_details.event_count** (*integer*) –
- **transactions[]**.**simple_hash_details.hash_schema_version** (*integer*) –
- **transactions[]**.**simple_hash_details.start_time** (*string*) –
- **transactions[]**.**transaction.block_number** (*string*) – Block number
- **transactions[]**.**transaction.blockhash** (*string*) – Block Hash
- **transactions[]**.**transaction.from** (*string*) – From...
- **transactions[]**.**transaction.gas** (*string*) – Gas left
- **transactions[]**.**transaction.gas_price** (*string*) – Gas Price
- **transactions[]**.**transaction.hash** (*string*) – Transaction hash
- **transactions[]**.**transaction.input** (*string*) – Input...
- **transactions[]**.**transaction.nonce** (*string*) – Nonce
- **transactions[]**.**transaction.r** (*string*) – Signature: R value
- **transactions[]**.**transaction.s** (*string*) – Signature: S Value

- `transactions[].transaction.to(string)` – To...
- `transactions[].transaction.transaction_index(integer)` – Transaction Index
- `transactions[].transaction.v(string)` – Signature: V value
- `transactions[].transaction.value(string)` – Value

GET /archivist/v1alpha2/blockchain:openapi

Get OpenAPI spec for Blockchain

Get OpenAPI v2.0 spec for Blockchain

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

GET /archivist/v1alpha2/blockchain:openapi-ui

Get OpenAPI UI for Blockchain

Get OpenAPI v2.0 UI for Blockchain

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

SYSTEM API

The system API allows access to the various resources in the Archivist system.

15.1 Archivistnode

Create the **bearer_token** and store in a file in a secure local directory with 0600 permissions.

See *API Request Authorization and Authentication*.

Note: The following example shows use of the API over curl in a bash terminal. The concepts are fully portable to any other REST client (eg PostMan or python requests)

Set the URL (for example):

```
$ export URL=https://symsation.1234-5678.nodes.archivist.jitsuin.io
```

The **archivistnode** endpoint reports on the status of the blockchain.

Query the endpoint:

```
$ curl -v -X GET \
  -H "@$BEARER_TOKEN_FILE" \
  $URL/archivist/v1/archivistnode
```

The response is:

```
{
  "identity": "quorum",
  "blockchain_nodes": [
    {
      "validator_pubkey": {
        "kty": "EC",
        "crv": "P-256K",
        "x": "VBKHictTWJC-3sqknXCb8MI4IxTc3c_My7lnem2C74E=",
        "y": "ItNeb5d-6vEHkvtUOcDYrEADxsZXeOCJm18pQWntenE=",
        "d": ""
      },
      "block_height": "38773",
      "connection_status": "REACHABLE",
      "genesis_hash":
        ↪ "0x1b526bd9c7f9bf7c43ba91ad07e5530eb7ceedf390396f9fbfeb68722e097e95",
      "state_root":
        ↪ "0x9606fc44a382938703678ac90581ab1260c9efd20ea8c7f90c87852bc982f3a7", (continues on next page)
```

(continued from previous page)

```

    "timestamp_committed": "2019-01-02T01:03:07Z",
    "timestamp_created": "2019-01-01T12:00:27Z",
    "syncing": null,
    "peers": [
      {
        "validator_pubkey": {
          "kty": "EC",
          "crv": "P-256K",
          "x": "o0uZ8ix5DE42srPCw1o22wYibkHGkvyCuLVqwcVAxb0=",
          "y": "W43sUjWg-ociR2x3CcAlWeOqc6oDkYuilJLup1q-ojU=",
          "d": ""
        },
        "connection_status": "REACHABLE"
      },
      {
        "validator_pubkey": {
          "kty": "EC",
          "crv": "P-256K",
          "x": "5HcU1PJgTe0LGyGxKFrIPFZWdTbxPySfi6bKxdQe08A=",
          "y": "dEpMURyTwEGzpgIgLdm4Csl1BgF6H39tb1Kf8wLLhVI=",
          "d": ""
        },
        "connection_status": "REACHABLE"
      }
    ]
  }
}

```

Note:

identity identifies the blockchain service.

The response contains a list of blockchain identities and attributes.

Each member of the list has the following attributes:

validator_pubkey public key (ECDSA).

block_height current no. of blocks in blockchain. May be zero if node is UNREACHABLE.

connection_status

- REACHABLE: node is reachable.
- UNREACHABLE: node is currently unreachable.

genesis_hash hash of the genesis block - use to verify that the blockchain is unchanged

state_root state_root for the public state in the genesis block - use to verify that the blockchain is unchanged

timestamp_committed timestamp (UTC) of latest block in the blockchain. May be zero (Jan 1st 1970) if node is UNREACHABLE.

timestamp_created timestamp (UTC) of genesis block in blockchain.

syncing if not null contains 3 integer fields, StartingBlock, CurrentBlock and HighestBlock indicating the the progress of syncing with the blockchain.

peers list of peers.

Each peer contains:

validator_pubkey public key (ECDSA).

connection_status

- REACHABLE: blockchain is connected.
- UNREACHABLE: peer is currently unreachable.

See [Swagger GET API](#)

15.2 Archivistnode Swagger API

GET /archivist/v1/archivistnode
Get information about an archivist node

Returns the identified archivist node

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **403 Forbidden** – Returned when the user is not authorized to read the archivist node's information
- **404 Not Found** – Returned when the identified archivist node does not exist
- **429 Too Many Requests** – Returned when a user exceeds their subscription's rate limit for requests.
- **default** – An unexpected error response.

Response JSON Object

- **blockchain_nodes[] .block_height** (*string*) – For the local node, the height of the last committed block (read only)
- **blockchain_nodes[] .connection_status** (*string*) –
- **blockchain_nodes[] .genesis_hash** (*string*) – HASH of genesis block. (read only)
- **blockchain_nodes[] .peers[] .connection_status** (*string*) –
- **blockchain_nodes[] .peers[] .validator_pubkey .keyEC .crv** (*string*) – The curve name for the key. 6.2.1.1 rfc 7518, 3.1 draft-ietf-cose-webauthn-algorithms-00 MUST be one of P-256K, P-256, P-384, P-521
- **blockchain_nodes[] .peers[] .validator_pubkey .keyEC .d** (*string*) – Present only for PRIVATE keys. The base64url encoding of the private key 6.2.2.1 rfc 7518
- **blockchain_nodes[] .peers[] .validator_pubkey .keyEC .kty** (*string*) –
- **blockchain_nodes[] .peers[] .validator_pubkey .keyEC .x** (*string*) – The base64url encoding of the x component for the uncompressed point. 6.2.1.2 rfc 7518
- **blockchain_nodes[] .peers[] .validator_pubkey .keyEC .y** (*string*) – The base64url encoding of the y component the uncompressed point 6.2.1.3 rfc 7518

- **blockchain_nodes[] .state_root** (*string*) – state root of genesis block. (read only)
- **blockchain_nodes[] .syncStatus.current_block** (*string*) – When syncing, the current block. (read only)
- **blockchain_nodes[] .syncStatus.highest_block** (*string*) – When syncing, the highest block. (read only)
- **blockchain_nodes[] .syncStatus.starting_block** (*string*) – When syncing, the starting block. (read only)
- **blockchain_nodes[] .timestamp_committed** (*string*) – Timestamp when consensus was achieved for the block at *block_height* (RFC 3339) obtained from the block info of the ‘latest’ block. (read only)
- **blockchain_nodes[] .timestamp_created** (*string*) – Timestamp of creation of the genesis block (RFC 3339) (read only)
- **blockchain_nodes[] .validator_pubkey.keyEC.crv** (*string*) – The curve name for the key. 6.2.1.1 rfc 7518, 3.1 draft-ietf-cose-webauthn-algorithms-00 MUST be one of P-256K, P-256, P-384, P-521
- **blockchain_nodes[] .validator_pubkey.keyEC.d** (*string*) – Present only for PRIVATE keys. The base64url encoding of the private key 6.2.2.1 rfc 7518
- **blockchain_nodes[] .validator_pubkey.keyEC.kty** (*string*) –
- **blockchain_nodes[] .validator_pubkey.keyEC.x** (*string*) – The base64url encoding of the x component for the uncompressed point. 6.2.1.2 rfc 7518
- **blockchain_nodes[] .validator_pubkey.keyEC.y** (*string*) – The base64url encoding of the y component the uncompressed point 6.2.1.3 rfc 7518
- **identity** (*string*) – The identity of the archivistnode blockchain (read only)

GET /archivist/v1/archivistnode:openapi
Get OpenAPI spec for Archivistnode

Get OpenAPI v2.0 spec for Archivistnode

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription’s rate limit for requests.
- **default** – An unexpected error response.

GET /archivist/v1/archivistnode:openapi-ui
Get OpenAPI html for Archivistnode

Get OpenAPI v2.0 html for Archivistnode

Status Codes

- **200 OK** – A successful response.
- **401 Unauthorized** – Returned when the user is not authenticated to the system.
- **429 Too Many Requests** – Returned when a user exceeds their subscription’s rate limit for requests.

- **default** – An unexpected error response.

MISCELLANEOUS

16.1 API Request Paging

The varied endpoints of the archivist API contain GET operations that return a `list` of json records. This list size is restricted at deployment of archivist (depending on the endpoint) to prevent buffer and memory overflow.

If the user wishes to access all records of the endpoint then the `page_size` parameter must be specified and the code must loop through repeated requests until no more records can be returned.

The first call to the endpoint specifies the `page_size`. The response body may contain a field `next_page_token`. If this is specified and non-empty repeated calls to the endpoint specifying `page_token=${next_page_token}` are made until no `next_page_token` (or an empty value) is returned in the response.

16.1.1 CURL

Specify the required `page_size` on the first call to the endpoint and then the `page_token` if specified in the response:

```
RESPONSE=$( curl -v -X GET \
    -H "@$BEARER_TOKEN_FILE" \
    $URL/archivist/v2/assets?page_size=100)
echo $RESPONSE | jq '.' # send to stdout

TOKEN=$( echo ${RESPONSE} | jq -r '.next_page_token' )

while [ -n "${TOKEN}" ]
do
    RESPONSE=$( curl -v -X GET \
        -H "@$BEARER_TOKEN_FILE" \
        $URL/archivist/v2/assets?page_token=${TOKEN})
    echo $RESPONSE | jq '.' # send to stdout

    TOKEN=$( echo ${RESPONSE} | jq -r '.next_page_token' )
done
```

16.1.2 Python

Alternatively a better solution is to use a python iterator. See the iterator method of the python class below.

```
"""Archivist interface (python 3.6)
"""

import json
import logging
from os import environ
from os.path import sep as ospathsep
import requests
import urllib3

class ArchivistException(Exception):
    """Indicate error
    """

class Archivist:

    def __init__(self, url, auth, root="archivist", verify=True):
        self._baseurl = url
        self._auth = 'Bearer ' + auth.strip()
        self._auth_headers = {
            'content-type': 'application/json',
            'authorization': self._auth,
        }
        self._root = root
        self._verify = verify

    @property
    def auth_headers(self):
        return self._auth_headers

    @property
    def baseurl(self):
        return self._baseurl

    @property
    def root(self):
        return self._root

    @property
    def verify(self):
        return self._verify

    def query_path(self, target):
        query_path = ospathsep.join([self._baseurl, self._root, target])
        logging.debug(query_path)
        return query_path

    def get(self, target):

        response = requests.get(
            self.query_path(target),
```

(continues on next page)

(continued from previous page)

```

        headers=self._auth_headers,
        verify=self._verify,
    )

    logging.debug(response)

    return response

def asset(self, asset_id):
    """Returns asset

    asset_id: asset identity
    """
    response = self.get(f"v2/{asset_id}")
    if response.status_code != 200:
        raise ArchivistException(f"Response code is {response.status_code}")

    return response.json()

def post(self, target, request):

    response = requests.post(
        self.query_path(target),
        data=json.dumps(request),
        headers=self._auth_headers,
        verify=self._verify,
    )

    logging.debug(response)

    return response

def iterator(self, path, field, page_size=None, query_params=""):
    """Returns iterator that lists objects

    path: REST path e.g. v2/assets
    query_params: REST query string - must begin with &

    If page size is specified return the list of records in batches of page_size
    until next_page_token in response is null.

    If page size is unspecified return up to the internal limit of records.
    (different for each endpoint)
    """

    paging = ""
    if page_size is not None:
        paging = f"page_size={page_size}"

    while True:
        response = self.get(f"{path}?{paging}{query_params}")
        logging.debug(response)

        if response.status_code != 200:
            raise ArchivistException(f"Response code is {response.status_code}")

        data = response.json()

```

(continues on next page)

(continued from previous page)

```

    try:
        records = data[field]
    except KeyError:
        raise ArchivistException(f"No {field} found")

    for record in records:
        yield record

    token = data.get("next_page_token")
    if not token:
        break

    paging = f"page_token={token}"

def assets(self, page_size=None, query_params=None):
    """Returns iterator that lists assets

    query_params: REST query string
    """
    return self.iterator(
        "v2/assets",
        "assets",
        page_size=page_size,
        query_params,
    )

def events(self, page_size=None, query_params=None):
    """Returns iterator that lists events

    query_params: REST query string
    """
    return self.iterator(
        "v2/assets/-/events",
        "events",
        page_size=page_size,
        query_params,
    )

def blockchain(self, identity, page_size=None):
    """Returns iterator that lists blockchain transactions

    identity: asset/event identity
    """
    return self.iterator(
        f"vialphal/blockchain/{identity}",
        "transactions",
        page_size=page_size,
    )

```

and typical usage may look like this:

```

# Initialise connection to Archivist
archivist = Archivist(archivist_url, authtoken)
if archivist is None:
    raise SystemExit

no_of_events = 0

```

(continues on next page)

(continued from previous page)

```
for event in archivist.events(
    page_size=10,
    query_params="confirmation_status=CONFIRMED",
):
    no_of_events += 1
    print("event", event)

no_of_transactions = 0
for transaction in archivist.blockchain(
    "assets/add30235-1424-4fda-840a-d5ef82c4c96f/events/11bf5b37-e0b8-42e0-8dcf-
    ↪dc8c4aefc000",
    page_size=10,
):
    no_of_transactions += 1
    print("transaction", transaction)
```


HTTP ROUTING TABLE

/archivist

	53	
GET /archivist/iam/v1/access_policies,	66	GET /archivist/v1/tlscacertificates, 37
GET /archivist/iam/v1/access_policies/{uuid},	68	GET /archivist/v1/tlscacertificates/{uuid}, 38
GET /archivist/iam/v1/access_policies/{uuid}/assets,	70	GET /archivist/v1/tlscacertificates:caps, 40
GET /archivist/iam/v1/access_policies:caps,	71	GET /archivist/v1/tlscacertificates:openapi, 40
GET /archivist/iam/v1/access_policies:openapi,	73	GET /archivist/v1/tlscacertificates:openapi-ui, 41
GET /archivist/iam/v1/access_policies:openapi-ui,	73	GET /archivist/v1alpha2/blockchain/assets/{asset_u
GET /archivist/iam/v1/assets/{uuid}/access_policies,	72	GET /archivist/v1alpha2/blockchain:openapi, 128
GET /archivist/iam/v1/subjects, 78		GET /archivist/v1alpha2/blockchain:openapi-ui, 130
GET /archivist/iam/v1/subjects/{uuid},	80	GET /archivist/v2/assets, 114
GET /archivist/iam/v1/subjects:openapi,	82	GET /archivist/v2/assets/{asset_uuid}/events/{uuid}, 124
GET /archivist/iam/v1/subjects:openapi-ui,	82	GET /archivist/v2/assets/{asset_uuid}/events/{uuid}, 126
GET /archivist/v1/archivistnode, 133		GET /archivist/v2/assets/{asset_uuid}/events/{uuid}, 116
GET /archivist/v1/archivistnode:openapi,	134	GET /archivist/v2/assets/{asset_uuid}/events, 122
GET /archivist/v1/archivistnode:openapi-ui,	134	GET /archivist/v2/locations, 94
GET /archivist/v1/attachments/{uuid},	84	GET /archivist/v2/locations/{uuid}, 95
GET /archivist/v1/attachments/{uuid}/info,	85	GET /archivist/v2/locations/{uuid}/permissions, 97
GET /archivist/v1/compliance/assets/{uuid},	48	GET /archivist/v2/locations:caps, 98
GET /archivist/v1/compliance_policies,	49	GET /archivist/v2/locations:openapi, 98
GET /archivist/v1/compliance_policies/{uuid},	51	GET /archivist/v2/locations:openapi-ui, 99
GET /archivist/v1/compliance_policies:caps,	52	POST /archivist/iam/v1/access_policies, 67
GET /archivist/v1/compliance_policies:openapi,	52	POST /archivist/iam/v1/subjects, 79
GET /archivist/v1/compliance_policies:openapi-ui,		POST /archivist/v1/compliance_policies, 50
		POST /archivist/v1/tlscacertificates, 38
		POST /archivist/v2/assets, 115
		POST /archivist/v2/locations, 94
		DELETE /archivist/iam/v1/access_policies/{uuid},

```
69
DELETE /archivist/iam/v1/subjects/{uuid},
80
DELETE /archivist/v1/compliance_policies/{uuid},
51
DELETE /archivist/v1/tlscacertificates/{uuid},
39
DELETE /archivist/v2/locations/{uuid},
96
PATCH /archivist/iam/v1/access_policies/{uuid},
69
PATCH /archivist/iam/v1/subjects/{uuid},
81
PATCH /archivist/v1/tlscacertificates/{uuid},
39
PATCH /archivist/v2/locations/{uuid},
96
PATCH /archivist/v2/locations/{uuid}/permissions,
98
```