retryHYCU R-Cloud Developer Guide

Revision: 18c4a396fbb

September 16, 2024

Table of Contents

[Summary 1](#_Toc178609369)

[Overview 1](#_Toc178609370)

[Getting started 1](#_Toc178609371)

[API compatibility statement 1](#_Toc178609372)

[Module run-time environment 1](#_Toc178609373)

[API categories 1](#_Toc178609374)

[Service configuration 1](#_Toc178609375)

[Synchronization 1](#_Toc178609376)

[Automatic policy assignment 1](#_Toc178609377)

[Backup configuration 1](#_Toc178609378)

[Location awareness 1](#_Toc178609379)

[Backup 1](#_Toc178609380)

[Restore 1](#_Toc178609381)

[Anomaly detection 1](#_Toc178609382)

[API usage patterns 1](#_Toc178609383)

[SaaS authentication 1](#_Toc178609384)

[Backup storage 1](#_Toc178609385)

[Return codes 1](#_Toc178609386)

[Error propagation 1](#_Toc178609387)

[Logging 1](#_Toc178609388)

[Module-specific context 1](#_Toc178609389)

[Data consistency 1](#_Toc178609390)

[Mode of operation 1](#_Toc178609391)

[Pagination 1](#_Toc178609392)

[Resource ordering semantics 1](#_Toc178609393)

[Resource restore dependencies 1](#_Toc178609394)

[Cross-source restore 1](#_Toc178609395)

[Consumption metrics for billing 1](#_Toc178609396)

[State transfer 1](#_Toc178609397)

[Bulk operations 1](#_Toc178609398)

[Module HTTP API reference 1](#_Toc178609399)

[GET /api/version - retrieve module manifest 1](#_Toc178609400)

[GET /api/schemas/auth - retrieve authentication schema 1](#_Toc178609401)

[POST /api/roots - list root resources 1](#_Toc178609402)

[POST /api/schemas/auth/validate - optional validation of custom authentication data 1](#_Toc178609403)

[POST /api/config/backup - optional Backup configuration for a resource 1](#_Toc178609404)

[POST /api/resources - list sub-resources 1](#_Toc178609405)

[POST /api/backup - backup a (sub)resource 1](#_Toc178609406)

[POST /api/bulk/backup - backup entire backupable 1](#_Toc178609407)

[POST /api/scenarios/restore - list possible restore scenarios 1](#_Toc178609408)

[POST /api/config/restore - restore options 1](#_Toc178609409)

[POST /api/restore - restore a (sub)resource 1](#_Toc178609410)

[POST /api/bulk/restore - restore entire backupable 1](#_Toc178609411)

[POST /api/status - get status of backup/restore operation 1](#_Toc178609412)

[POST /api/bulk/status - get status of bulk backup/restore operation 1](#_Toc178609413)

[POST /api/abort 1](#_Toc178609414)

[POST /api/retention/cleanup - remote storage cleanup 1](#_Toc178609415)

[Storage Service HTTP API reference 1](#_Toc178609416)

[Error handling 1](#_Toc178609417)

[POST /api/v2/objects/{objectPath} - upload object 1](#_Toc178609418)

[GET /api/v2/objects/{objectPath} - download object 1](#_Toc178609419)

[Language bindings (SDK) 1](#_Toc178609420)

[Layout schema definition 1](#_Toc178609421)

[Layout positioning elements 1](#_Toc178609422)

[Layout control elements 1](#_Toc178609423)

[Layout helper elements 1](#_Toc178609424)

[Rules: 1](#_Toc178609425)

[Payload: 1](#_Toc178609426)

[Full JSON Schema example: 1](#_Toc178609427)

## Summary

This document describes the HYCU R-Cloud module API and development-related procedures. This document is intended for module developers. Readers’ familiarity with HTTP/REST APIs and JSON is assumed.

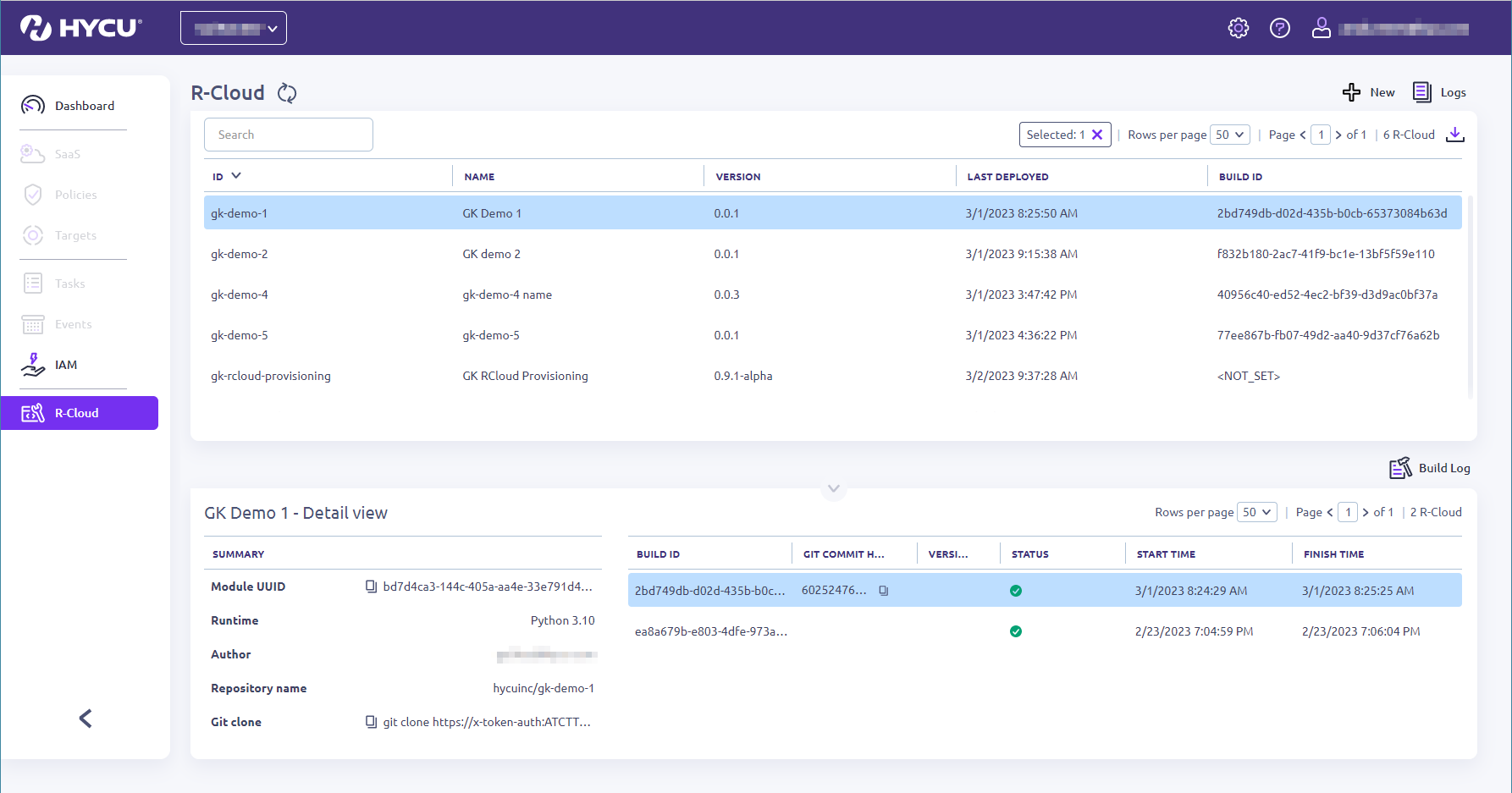
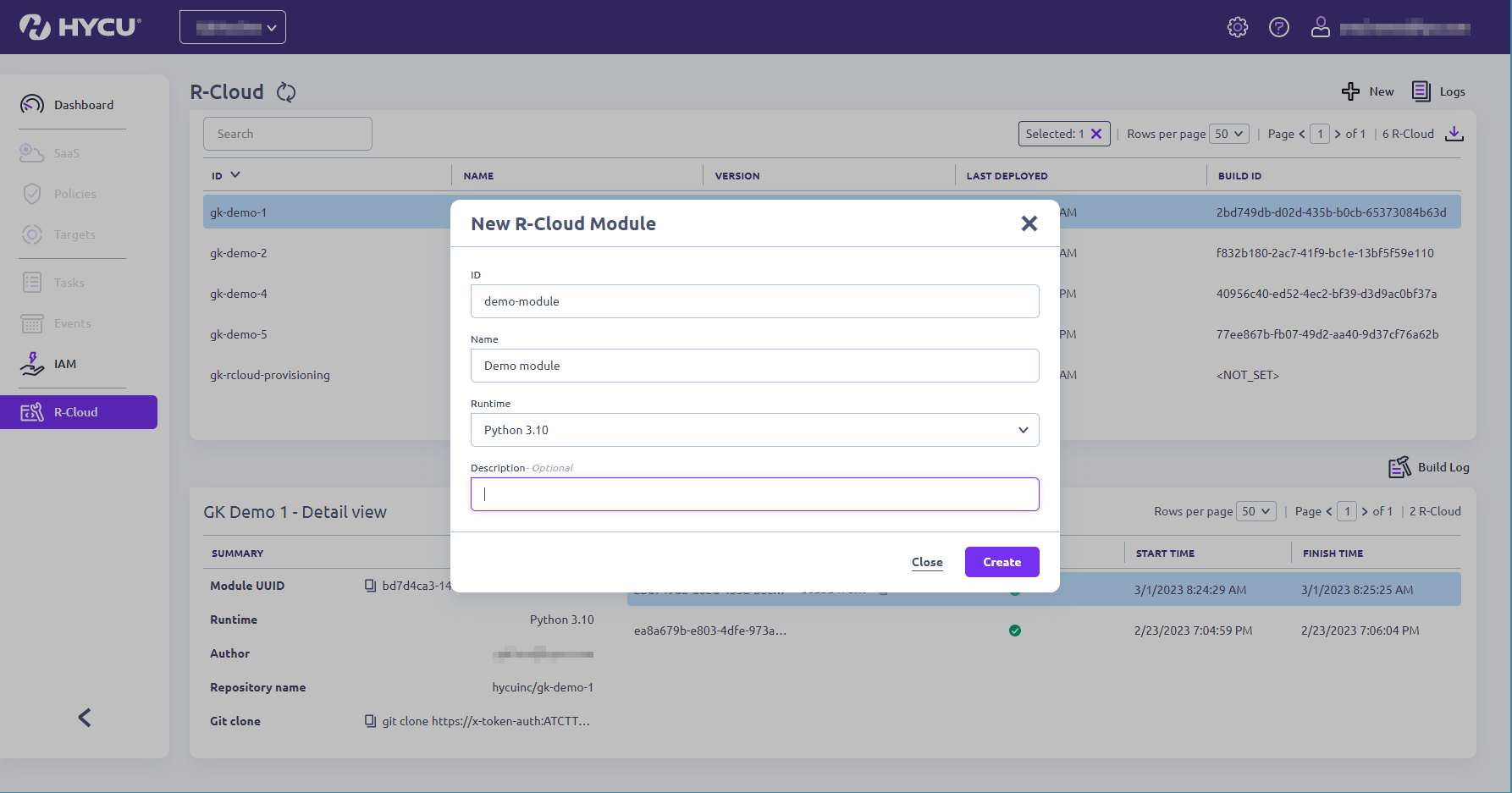
## Overview

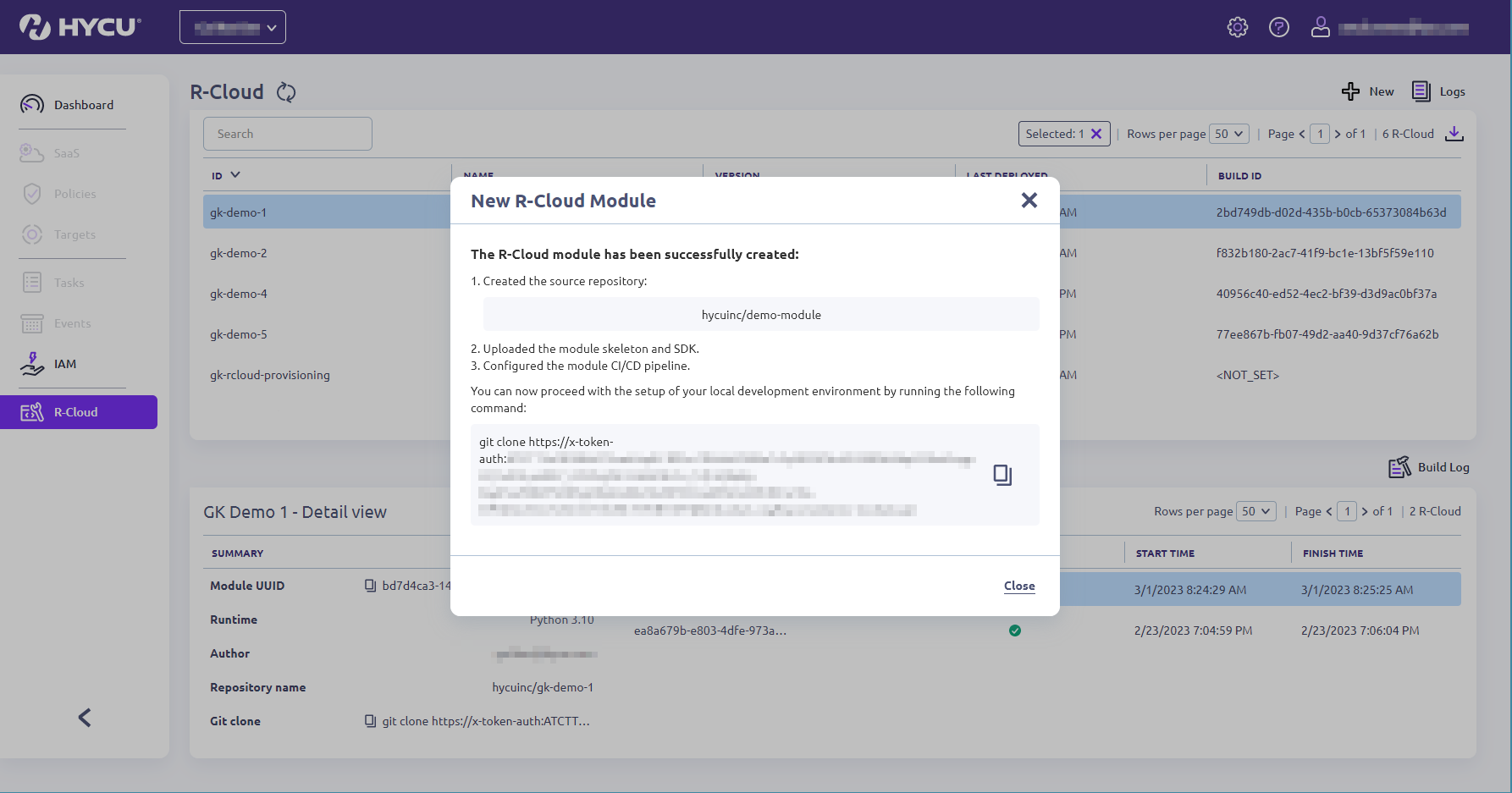
HYCU Protégé is cloud data protection platform that protects cloud workloads, such as virtual machines, object storage buckets and Kubernetes clusters. Data protection is driven by a user-defined policy that describes service level agreements such as recovery-point objective (RPO), data retention and redundancy, and object storage location where to store the data. HYCU Protégé protects data according to the policy and provides policy compliance information and real-time progress updates.

HYCU R-Cloud extends HYCU Protégé with a pluggable modular architecture to enable backup and recovery of arbitrary cloud services, and resources to accelerate development of modules.

## Getting started

To create a new module:

* Log into your HYCU R-Cloud developer console.
* Select ‘Subscription’ in the drop down menu at the top of the console.
* Select ‘R-Cloud’ panel from the navigation menu on the left side of the console. 
* Click ‘New’ button. 
* Enter the needed information in the dialog (ID, module name, runtime, description), and click ‘OK’.

HYCU will provision a git repository containing code for a sample module, build and deploy the initial version of the module, and display instructions to clone the code repository. 

Clone the code repository and follow the guidance provided in the README.md to add cloud service-specific functionality and test it locally. When you commit and push your code changes, HYCU will automatically build, deploy and make the changes available for use in HYCU R-Cloud.

You may use the “Modules” panel to see the list of deployed modules, their build status and logs, and inspect the module logs.

A deployed module can be configured in the “Sources” menu. After this step, HYCU R-Cloud can protect cloud service resources provided by the module.

## API compatibility statement

During technical preview, HYCU does not assure API backward compatibility. Partners will be notified of APIs that change. After general release, API will be assured to be backward compatible, unless notified ahead of time:

* Current endpoints and payloads may be extended in a way that does not break backward compatibility. Therefore, a module endpoint must accept unknown fields in a request.
* New endpoints may be added.

## Module run-time environment

The HYCU R-Cloud modules are packaged as containers deployed by R-Cloud on-demand and as required.

There is a 60 minute limit on request execution. Module can be either stateless or stateful. Stateless modules should not expect to continue with the execution after the request returns its result. Stateful modules will retain internal state for the duration of the backup or restore. For details, see “Bulk operations”.

HYCU services interact with the module via its HTTP API. Authentication and TLS termination are handled by the serverless platform and automatically set up during the module deployment.

## API categories

For each service that requires data protection, HYCU will call the module APIs for:

* Service configuration
* Synchronization of backupables
* Backup configuration
* Backup of service data
* Restore option definition
* Restore of service data

The modules serve an HTTP API corresponding to the following operations:

* Retrieve module manifest
* Retrieve schema information for authentication, backup and restore configuration
* List root resource (backupable)
* List resources of a root
* (Start) backup of a resource
* (Start) restore of a resource
* Retrieve status of a backup/restore

Request and response bodies are JSON documents containing:

* Fields generated or parsed by HYCU
* (Opaque) fields known only to the module - these are not interpreted by HYCU.

To allow module schema evolution, the HYCU R-Cloud modules should tolerate and ignore any unknown fields in requests.

### Service configuration

To interact with a SaaS application, the module may need to be configured to authenticate with the SaaS service. Since the authentication mechanisms differ between SaaS services, the module API may define an authentication schema which would for example define the required fields.

In HYCU, a configured instance of a module is called a “source”. A user may define multiple sources for each module for example, to protect different instances of a SaaS service).

During source configuration, HYCU will retrieve the module manifest and schema information. This would for example allow HYCU to display a module-defined UI dialog to enter authentication details, validate the inputs (if applicable) and save the output of the resulting interaction in its database.

The saved authentication information is sent with each module API request. For details, see “SaaS authentication”.

### Synchronization

To protect SaaS data, HYCU must be aware of the objects that the module offers for protection (i.e., backupables).

After a source is configured and added, HYCU will periodically perform a “synchronization”, where it will list root resources of all configured module sources. Output will be stored in the HYCU database and relevant parts displayed in the SaaS panel of the HYCU UI, where the user may apply a policy to start protecting these SaaS backupables.

The module should provide a stable identifier for each backupable (for example, to be able to track its existence across renames). This identifier should be unique within the source.

### Automatic policy assignment

Result of /api/roots may return a resources[].labels field that describes list of arbitrary labels defined in the SaaS application itself (if applicable). This can be used by HYCU to automatically assign policies during synchronization. For example, a policy with defined labels with key “my-label” and value “my-value” will be assigned to a resource that returns a label with key “my-label” and value “my-value”.

This can be used to influence HYCU policy assignment from the SaaS application itself for example, as part of DevOps automation.

For details, see ‘Setting up automatic policy assignment’ in HYCU Help.

### Backup configuration

The module may implement /api/config/backup to define configuration options for the backup of the root resources. In that case, user may select a resource and configure sub-resource exclusion and backup options.

### Location awareness

Result of /api/roots may return a resources[].location object that describes resource location in a public cloud (if applicable). HYCU may use this information to optimize costs for example, by avoiding cross-region data transfer.

### Backup

HYCU policy describes the service-level agreements (SLA) for protection for example, the recovery point objective (RPO) which describes how frequently a backupable needs to be backed up.

Once the HYCU scheduler determines that a backupable should be backed up, it will start a backup task by performing the following steps:

1. (Recursively) call /api/resources of the backupable.
2. Call /api/backup for each sub-resource.
3. (Async-only) call /api/status for every sub-resource until its backup is completed.

Backup completes after all listed sub-resources are backed up. HYCU stores listed sub-resources in a backup catalog, to use them during a restore.

If /api/resources returns with a non-null errors field:

* backup task is marked as “completed with errors”

If /api/backup or /api/status response contains a non-null errors field:

* If status is ERROR, backup task is marked as “completed with errors”. All sub-resources of the failing resource are skipped.
* If status is DONE, backup task is marked as “completed with warnings”.

If backups of all (sub)resources fail with ERROR status, entire backup task fails with ERROR status. All errors are logged. Up to 20 errors per step are added to the task report.

Any request that returns 4XX or 5XX HTTP code terminates the backup immediately.

### Restore

When a user selects restore action on the specific backup, HYCU UI shows Restore dialog where the user is guided through the following wizard steps:

1. The user selects type of resource to restore (scenario).
   * The module may implement /api/scenarios/restore endpoint to provide detailed description for the possible scenarios.
2. The user selects resource(s) to restore.
3. (Optional) The user specifies restore options for the selected type of resource.
   * The module may implement /api/config/restore endpoint to define available restore options.

On restore confirmation, HYCU retrieves the backup catalog and creates a list of resources to restore. It then performs the following steps:

1. Call /api/restore for each sub-resource.
2. (Async-only) call /api/status for every sub-resource until restore is completed.

If /api/restore or /api/status response contains a non-null errors field:

* If status is ERROR, restore task is marked as “completed with errors”.
* If status is DONE, restore task is marked as “completed with warnings”.

Any request that returns 4XX or 5XX HTTP code terminates the restore immediately.

### Anomaly detection

HYCU offers anomaly detection functionality that compares state of current backup with the previous backup and displays a summary of changes in the task report. Anomaly detection can be used to detect unauthorized changes to the SaaS application. This may not apply to all SaaS applications, so module should indicate it supports this functionality by setting feature flag ANOMALY\_DETECTION.

## API usage patterns

### SaaS authentication

Modules can have multiple authentication methods. The module communicates the actual authentication requirements via the manifest returned by /api/version and authentication schema returned by /api/schemas/auth.

#### Available authentication methods

The following authentication methods are available:

1. AUTH\_GC\_ACCESS\_TOKEN

This authentication method should be used by modules that require access to Google Cloud services. With this mode, HYCU will generate an access token for a selected Google Cloud service account (or HYCU-managed service account, if selected) and pass it with every authenticated request.

1. AUTH\_AWS\_SESSION\_TOKEN

This authentication method should be used by modules that require access to Amazon AWS services. With this mode, HYCU will generate a session token and pass it with every authenticated request.

1. AUTH\_SETTINGS

This authentication method should be used for SaaS applications that have their own authentication mechanisms, for example API keys. When using this authentication method, module must define a schema for settings by implementing /api/schemas/auth endpoint. HYCU uses this schema to generate a UI dialog for configuration of authentication. Module may define /api/schemas/auth/validate endpoint for validation of the payload.

#### Passing authentication data to the module

Most requests to the module API accept an authInfo field, containing authentication information.

The field authInfo contains the authentication information for the module. Content depends on the auth capability and schema:

1. AUTH\_AWS\_SESSION\_TOKEN

Field authInfo contains:

{  
 "awsSessionToken": {  
 "accessKeyId": "string" // access key identifier  
 "secretAccessKey": "string" // secret access key  
 "sessionToken": "string" // session token  
 }  
}

1. AUTH\_GC\_ACCESS\_TOKEN

Field authInfo contains:

{  
 "gcAccessToken": {   
 "accessToken" : "string" // gc access token value  
 }  
}

1. AUTH\_SETTINGS

The field authInfo contains:

{  
 "settings": {  
 // value of payload object from /api/schemas/auth  
 ...  
 }  
}

For example, a schema that contains:

{  
 ...  
 "payload": {  
 "mySettings": {  
 "user": "usernameFieldId",  
 "key": "passFieldId"  
 }  
 }  
}

would persist the following object in HYCU:

{  
 "mySettings": {  
 "user": "usernameFieldId",  
 "key": "passFieldId"  
 }  
}

and send the following part of the request body to the plugin:

{  
 ...  
 "authInfo": {  
 "settings": {  
 "mySettings": {  
 "user": "usernameFieldId",  
 "key": "passFieldId"  
 }   
 }  
 }  
 ...  
}

For details, see GET /api/schemas/auth.

### Backup storage

Module can read or write its data to and from different storage backends. Multiple object storage backends are available: S3, Google Cloud Storage and a backend-agnostic HYCU storage service. Module specifies which storage types it can support using the field capabilities.supportedStorageTypes in the response of the /api/version endpoint:

* STORAGE\_GC: Google Cloud Storage (deprecated)
* STORAGE\_S3: AWS S3 or S3-compatible (deprecated)
* STORAGE\_SERVICE: HYCU Storage Service

Deprecation notice: STORAGE\_GC and STORAGE\_S3 are deprecated. New modules should use STORAGE\_SERVICE.

#### Backup storage authentication

To handle different storage backends, HYCU sends storage authentication with /api/backup, /api/restore and /api/status requests in the storage field:

{  
 ...  
 "storage": {  
 "type" : "<storage type>", // string; one of: STORAGE\_GC, STORAGE\_S3, STORAGE\_SERVICE  
 "authInfo": { // type-specific authentication  
 // STORAGE\_S3: see AUTH\_AWS\_SESSION\_TOKEN   
 // STORAGE\_GC: see AUTH\_GC\_ACCESS\_TOKEN  
 // STORAGE\_SERVICE: empty  
 ...  
 },  
 "url": "<object path>", // storage-specific and operation-specific fully-qualified path the storage endpoint  
 }  
 ...  
}

* storage.url value is operation-specific:
  + For /api/backup, it contains a backend-specific path to the storage backend (same as now deprecated targetPath).
  + For /api/restore, it contains a backend-specific path to the returned workingPath in response of /api/backup or related /api/status.
  + For /api/status, it contains the same value as related /api/backup or /api/restore.

Non-empty values of authInfo fields use the same semantics as “SaaS authentication”.

### Return codes

* 200 OK indicates successful call.
* 4XX or 5XX indicates a fatal error. Fatal errors result in immediate termination of the operation.

### Error propagation

#### Fatal errors

For fatal errors (http status code 4xx or 5xx), response should contain a field error that contains a single error object:

{  
 "error": {  
 "code": 0, // integer, could be http code or any other meaningful status  
 "message": "error text" // human-readable error description  
 "payload": [ { object }, ...] // optional payload, may be absent. If present, value must not be null  
 },  
}

#### Non-fatal errors

For non-fatal errors (http status code 20X), all methods that accept authInfo may set a field errors in their response, that contains an array of error objects. Error descriptions may be shown in HYCU user interface, for example in the task reports. Presence of errors may influence task status.

{  
 ...  
 "errors": [  
 {  
 "code": 0, // integer, could be http code or any other meaningful status  
 "message": "error text" // human-readable error description  
 "payload": [ { object }, ...] // optional payload, may be absent. If present, value must not be null  
 },  
 ...  
 ],  
 ...  
}

### Logging

Anything module writes to stdout and stderr is logged. Module should mask or anonymize any sensitive data that is logged.

Timestamps are automatically added. [JSON structured logging](https://cloud.google.com/logging/docs/structured-logging) is recommended to enable log filtering by severity.

Example of a structured log entry:

{ "severity":"ERROR", message:"This is an error message" }

HYCU specifies the lowest severity level that module should log via HYCU\_LOGGING\_LEVEL environment variable. Possible values are DEBUG,INFO,WARNING,ERROR and CRITICAL.

### Module-specific context

Cloud services do not have a uniform model and API. To generalize this cloud service-specific behavior, HYCU module API accepts and returns “context” fields in request and response bodies. Content of the context is module-specific and not interpreted by HYCU (with some exceptions). This lets the module define variables relevant to the cloud service object model and operations (for example, identifiers, state).

Context fields may be persisted in the HYCU database. Module should not put sensitive data in the context. Context fields should contain *only* what is needed:

* for the backup phase to identify the resource under backup in the SaaS application, and
* for the restore phase to reliably identify the storage objects needed for restore.

### Data consistency

Cloud services may lack mechanisms to maintain consistency during backup for example, transactions or snapshots.

Due to sequencing of operations, there may be some time between /api/resources that produces the ctx and the /api/backup that consumes it. Module should handle any potential discrepancies.

Resources that existed during /api/resources may no longer exist when /api/backup is called. If this is expected, module should not treat this as a fatal error, and continue with backup of other resources.

Context information from time of /api/resources may not be up to date when /api/backup is called. Module should use ctx from /api/resources to identify the resource that needs to be backed up, and use ctx returned on the successful completion of /api/backup or /api/status to provide an updated context for recovery purposes.

### Mode of operation

Module may utilize synchronous or asynchronous backup methods.

Asynchronous methods may be used if cloud service has native jobs to aid backup and restore for example, a concept of an import or export job, or snapshot capability.

Synchronous methods may be used if cloud does not offer any native export/import job functionality for example, where backup is synthesized by utilizing the cloud service API to retrieve/crawl API objects.

### Pagination

The /api/roots and /api/resources endpoints support pagination.

Pagination is optional, but recommended. The module should use pagination to balance the number and duration of requests to the SaaS resources and the number of records received, to prevent excessive memory consumption and reduce the load on the SaaS application.

If pagination is used, module should return a response that sets field hasMoreRecords to true, and provides a paginationCtx field that can be used for the next request. HYCU will repeat the request with the received paginationCtx. This continues until the module returns a response with hasMoreRecords to false, or a fatal error.

When used, module should populate paginationCtx in the response with state that provides data to obtain the next batch of resources for example, the page number or link to the next page. This state transfer may also be used to handle for example, rate limiting.

### Resource ordering semantics

Module can influence backup and restore ordering using sequence groups, defined separately for backup and restore. Resources returned by /api/roots/ and /api/resources with the same sequence group may be processed in parallel (usually batched). Sequence groups are processed sequentially.

### Resource restore dependencies

There are situations where restore of a sub-resources depend on knowledge of the restore of the parent resource. An example is redirected restore, where sub-resource must be aware that parent resource is available under a different ID. To handle this, the restore operation passes the context of parent’s restore response in the parentCtx field of the restore request to parent’s sub-resources.

### Cross-source restore

Module may support restoring data to a different source for example, for restore testing or cloning. Module indicates support for such scenarios with the CROSS\_SOURCE\_RESTORE feature flag. Module documentation should state any limitations that apply to cross-source restore.

### Consumption metrics for billing

Module must publish consumption metrics that HYCU uses for billing for example, source capacity of a root resource, or number of users of a subscription. Metrics are reviewed by HYCU.

Note: Consumption metrics are not APM or statistical metrics. Consumption metrics are used to calculate billing.

Module sends metrics as part of /api/roots response. For details, see metrics field for root resource objects in /api/roots. A root resource can have multiple metrics.

Metrics are defined by three fields:

* ID: a string that identifies the metric.
* value: an integer or string value of the metric.
* scope: a string that identifies the scope of the metric.

Metrics that are related to a root resource use the scope that is same as the ID of the root resource. For clarity, scope value should be prefixed with the type of the entity that the scope is used.

Examples of scope: - For CloudSQL, metrics with ID com.hycu.allocatedSize describes CloudSQL instance size, so the scope is set to instance:<fully qualified instance identifier>. - For Jira modules, metrics with ID com.hycu.activeUsers represents the number of users of the Jira organization, so the scope is set to organization:<organization ID>.

ID of the metric should be unique to the concept that the metric communicates. Unit of the metric’s value should be defined by the metric and must not change.

Note that metric value can be string or integer. Integers should be used for cardinal numbers, and HYCU may apply operations on integer values such as sum, average or mean. If metric is a cardinal number, ensure that the value is not returned as a string.

Module may define own metrics. The following common metrics are defined:

* com.hycu.activeUsers: number of users of a service, subscription or instance
  + Example: Number of users of an Atlassian Jira organization
* com.hycu.allocatedSize: allocated size of root resource
  + Example: total space available to a database instance.
* com.hycu.usedSize: size of used data of a root resource.

#### Example metrics

**Google Cloud SQL**

* Backupable is a database instance uniquely identified with a fully qualified instance identifier.
* Id and value describe the allocated storage size of the instance.

{  
 "id": "com.hycu.allocatedSize",  
 "scope": "instance:projects/qa-project-227914/instances/saas-mysql80-niki-20231108-132157",  
 "value": 10737418240  
}

**Amazon DynamoDB**

* Backupable is a table uniquely identified by ARN (Amazon Resource Name).
* Id and value describe the size of the table.

{  
 "id: "com.hycu.usedSize",  
 "scope": "table:arn:aws:dynamodb:us-east-1:712244577098:table/tabela012",  
 "value": 14111511  
}

**Atlassian Jira Software**

* Backupable is a Jira Software project, billed by the number of users in the organization.
* Scope describes the organization that the Jira project is a member of.
* Value describes the the number of users for the organization’s Jira subscription.

{  
 "id": "com.hycu.activeUsers",  
 "scope":"organization:test-org.domain",  
 "value": 50  
}

### State transfer

State transfer via context between backup/restore/status and status module API calls can be used to handle different interactions with the SaaS API. We describe two common scenarios: rate-limiting and multiphase operations.

#### Scenario: rate limiting

SaaS API call fails with code 429 - Rate limit exceeded. The module may return with status PENDING or RUNNING and set a context that contains a timestamp of the failure, and the information when to retry. The next status call will pass this context, which the module may use to determine if it can retry the operation:

* if it cannot, it will return status “PENDING”
* if it can, it retries and eventually returns status “DONE”.

Note that there is a 60 minute limit on request execution.

#### Scenario: multiphase operations

Backup of one (sub)resource may be composed of multiple calls to the SaaS API. The module may return status “PENDING” and populate context with information of the current state/phase. Next status call may use this context to perform the next phase. This method can be applied to retries or potential post-failure cleanup.

### Bulk operations

Module can operate in a stateful mode, signaled with BULK\_OPERATIONS feature flag. This mode allows the module to better utilize SaaS application APIs, such as using bulk APIs, export-transform-load (ETL) workflows, and reusing state between API calls. However, it can significantly increase the complexity of the module.

When enabled, HYCU will call /api/bulk/backup and /api/bulk/restore endpoints to delegate backup of the entire SaaS backupable to the module. Module is expected to internally handle listing, parallel processing and catalog handling according to the defined schema to enable granular recovery scenarios.

HYCU monitors the status of the bulk operation using /api/status endpoint. HYCU may trigger an abort of the bulk operation using /api/abort endpoint. Use of the storage service is recommended for bulk operations. To facilitate this mode, HYCU will spawn a module process that will remain running for the duration of the entire backup or restore.

#### Bulk backup

1. HYCU calls the module’s /api/bulk/backup for the root resource of the backupable.
2. HYCU calls /api/status to track progress of the bulk restore.
3. After /api/status returns DONE, HYCU collects the browse catalog in a location shared with the module.

Summary of number of processed resources of each type should be reported via field summary in the /api/status response.

#### Bulk restore

1. HYCU retrieves and browses the catalog for the restore point.
2. HYCU calls the module’s /api/bulk/restore, providing the full catalog and the UI selection
3. HCU calls /api/status to track the progress of the bulk restore.

Summary of number of processed resources of each type should be reported via field summary in the /api/status response.

#### Error handling for bulk operations

Bulk operations shift the granular operations into the module. As a consequence, the standard non-fatal error handling is insufficient:

* Standard errors does not describe the resource that the error applies to.
* A single bulk operation may impact many resources, so a single bulk operation error may produce a large errors response, causing scalability issues.

To mitigate this scalability risk, bulk operations should store the error data in a separate table in the catalog database. Schema for this table is described in “Error schema”. This applies to both backup and restore. HYCU will process errors from this table to produce a sample for the task output.

## Module HTTP API reference

### GET /api/version - retrieve module manifest

This endpoint is used during module deployment and source configuration. Manifest describes general details of the module.

Response body:

{  
 "name": "string", // module name  
 "version": "string", // module version  
 "source": "string", // optional info about module source, e.g., git commit hash  
 "capabilities": {  
 "api": [ "v0",... ], // supported Module API versions  
 "auth": [ "string",... ], // string, any of: AUTH\_AWS\_SESSION\_TOKEN, AUTH\_GC\_ACCESS\_TOKEN, AUTH\_SETTINGS  
 "features": ["string",...], // array of strings, any of: REMOTE\_STORAGE\_ONLY, BULK\_OPERATIONS, CROSS\_SOURCE\_RESTORE, ANOMALY\_DETECTION  
 "supportedStorageTypes": [ "string", ... ] // enum: STORAGE\_GC, STORAGE\_S3, STORAGE\_SERVICE. Conflicts with features["REMOTE\_STORAGE\_ONLY"].  
 }  
}

The field capabilities is used to communicate to HYCU that the module supports specific behavior. The content of this field may be extended to describe additional behaviors when added to HYCU.

Current capabilities:

* api: array describing supported Module API versions (currently only “v0”)
* auth : array describing authentication supported by module
  + AUTH\_AWS\_SESSION\_TOKEN: support AWS session token authentication
  + AUTH\_GC\_ACCESS\_TOKEN: support GC access token authentication
  + AUTH\_SETTINGS: support custom authentication schema, see /api/schemas/auth
* features: array describing additional properties
  + REMOTE\_STORAGE\_ONLY: this flag indicates that the module uses only remote storage
  + BULK\_OPERATIONS: this flag indicates that module is capable of bulk operations
  + CROSS\_SOURCE\_RESTORE: this flag indicates that module supports restoring resources to a different source
  + ANOMALY\_DETECTION: this flag indicates that module supports the anomaly detection use case and does not require any additional storage for a backup. For example, the feature is usable if a module stores backup on remote snapshot storage.
* supportedStorageTypes: array describing storage types supported by the module.
  + The field should not be defined if the module declares REMOTE\_STORAGE\_ONLY feature.
  + This field deprecates use of targetPath field in /api/backup. Use of targetPath assumes previous default behavior (Google Cloud Storage with implicit credentials).

For details about authentication, see “SaaS authentication”. For details about storage types, see “Backup storage”.

### GET /api/schemas/auth - retrieve authentication schema

This endpoint is used during source configuration to retrieve the information about authentication requirements. This is called only if the module advertises AUTH\_SETTINGS auth capability.

Example response body:

{  
 "schemaType": "hycu.layout",  
 "schema": {  
 "layout": {  
 "elements": [  
 {  
 "type": "column",  
 "elements": [  
 {  
 "id": "instanceUrl",  
 "type": "text",  
 "properties": {  
 "label": "Url",  
 "placeholder": "url"  
 },  
 "validators": {  
 "required": "true"  
 }  
 },  
 {  
 "id": "instanceUser",  
 "type": "text",  
 "properties": {  
 "label": "User",  
 "placeholder": "user123"  
 },  
 "validators": {  
 "required": "true"  
 }  
 },  
 {  
 "id": "instancePassword",  
 "type": "text",  
 "properties": {  
 "label": "Password",  
 "inputType": "password"  
 },  
 "validators": {  
 "required": "true"  
 }  
 }  
 ]  
 }  
 ]  
 },  
 "payload": {  
 "instanceUrl": "instanceUrl",  
 "instanceUser": "instanceUser",  
 "instancePassword\_secret": "instancePassword"  
 }  
 }  
}

The field schemaType currently supports only one type: hycu.layout. This drives generation of a UI configuration dialog. On save, UI sends a JSON object described by payload to HYCU.

For details of the hycu.layout schema, see section “Layout schema definition”.

Additional schema types may be added in the future for example, to allow the module to define input validation rules and richer input type definitions (drop-downs, check-boxes, etc.).

### POST /api/roots - list root resources

This endpoint is called during synchronization. Responses are interpreted and persisted in the HYCU database.

Request body:

{  
 "filter": ["string",...],  
 "authInfo": {  
 ...  
 },  
 "paginationCtx": {   
 // optional; module-specific context for pagination from previous response.  
 // First request will always omit this field.  
 ...  
 }  
}

* The field paginationCtx is passed from previous call if it returned hasMoreRecords and paginationCtx. First request will always omit this field. For details, see “Pagination”.
* The field filter is an optional list of strings that the module should use to restrict the set of returned resources. Its contents depend on the module, for example, it could be a list of GCloud project IDs or similar. The field may be absent but not null. An empty list means that no filtering should be done.

Response body:

{  
 "resources": [  
 {  
 "id": "string", // unique identifier of the root resource within this source  
 "name": "string", // display name of the root resource  
 "type": "string", // type of the root resource  
 "canBackup": false, // boolean, true if resource supports backup  
 "backupSeqGroup": 1, // integer, sequence group number for backup  
 "restoreSeqGroup": 1, // integer, sequence group number for restore  
 "hasSubresources": true // boolean, true if resource can be expanded  
 "ctx": {   
 // key-value map, module-specific context - opaque to HYCU  
 },  
 "displayCtx": {  
 // optional, key-value map, information about the root resource for UI detail display  
 },  
 "labels": [  
 // optional, may contain an array of JSON objects with "key" and "value" fields  
 // The field is used for automatic backup policy assignment.  
 {  
 "key": "my-test-key1", // required, max length is 255  
 "value": "my-test-value" // required  
 }  
 ],  
 "location": { //optional, placeholder for root location information  
 "cloud": { //optional, set if root resource is located in supported cloud  
 "type": "GC", //required, currently supported values are "GC" or "AWS"  
   
 // optional region string in cloud specific format, max length 255 Unicode characters  
 // In case of a multi region resource:  
 // If possible, return a specific region that is appropriate for the backup location.  
 // If that's not available, but a multi-region identifier is (for example, "eu" or "us"), return the multi-region identifier.  
 // Otherwise, do not return the region field.  
 "region": "us-west1",   
  
 // mandatory for GC-only: project ID where the backupable resides.  
 "project": "project-id"  
 }  
 },  
 "metrics": [ // optional - metrics published by module  
 {  
 "id": "com.hycu.activeUsers", // ID of the metric  
 "value": 100, // value of the metric  
 "scope":"test-scope", // scope of the metric  
 },  
 ...  
 ]  
 }  
 ],  
 "hasMoreRecords": true, // optional; boolean, true if request should be repeated to obtain more data  
 "paginationCtx": { // optional; module-specific context for pagination. Should be set if hasMoreRecords is set to true.  
 }  
}

* The field id should be a unique identifier within the source.
* Field name is for display purposes and is considered volatile (may change between two calls).
* The field displayCtx can be used to describe details about the root resource. These are shown with minimal interpretation (pretty-printing of the key) in the HYCU UI in the details pane of the selected SaaS backupable.
* The field ctx should contain data that allows the module to unambiguously identify the SaaS object that a list/backup operation may apply to.
* The field metrics contains consumption metrics for the module. For details, see “Consumption metrics”.
* The field hasMoreRecords indicates the response is incomplete and should be repeated.
* The field paginationCtx lets the module provide the pagination state used for the next request.

### POST /api/schemas/auth/validate - optional validation of custom authentication data

This endpoint is called when a user saves the source configuration. This endpoint is optional. If the module does not support the endpoint (returns 404 or 501 HTTP status), the validation is not performed.

If the endpoint is implemented, it should validate the request for example, by connecting to the SaaS application.

* If validation fails, validation error message from the module is shown in a notification window and source configuration update is prevented.
* If validation succeeds or is not implemented, source configuration update proceeds.

Request body:

{  
 "authInfo": {  
 ... // authentication information to validate  
 }  
}

Response body:

If validation is successful, module returns a 200 OK with an empty object.

{  
}

If validation fails, module returns a fatal error (for example http code 403 or 500) as described in “Fatal errors”:

{  
 "error": {  
 "code": 0, // integer, could be http code or any other meaningful status  
 "message": "error text" // human-readable error description  
 "payload": [ { object }, ...] // optional payload, may be absent. If present, value must not be null  
 },  
}

### POST /api/config/backup - optional Backup configuration for a resource

This endpoint is called when a user selects Backup Configuration action for the specific backupable in HYCU UI.

The response is used for rendering the following tabs in the Backup Configuration dialog: \* Possible excludes - optional, hierarchical list of items that can be excluded from a backup. \* Backup options - optional, any additional options for a backup, that a user may configure.

Response is persisted in the HYCU database and passed as excludes and options fields to /api/backup calls for all sub-resources of the configured backupable.

This endpoint is optional. If the module does not support the endpoint (returns 404 or 501 HTTP status), the user is informed that there is no Backup Configuration is available for the specific backupable.

Request body:

{  
 "ctx": {  
 // resource[].ctx returned as response of /api/roots call  
 },  
 "authInfo": {  
 ...  
 }  
}

Response body:

{  
 "possibleExcludes": [  
 {  
 "ctx": {"myKey": "db1"}, // if the item is excluded, ctx is sent to the Module during backup process  
 "id": "db1", // id should be unique per backupable  
 "name": "db1", // used for rendering a list of the possible excludes in HYCU Backup Configuration dialog  
 "subResources": [ ... ] // sub-resources for this excluded resource  
 }  
 ],  
 "possibleOptions": {  
 "schemaType": "hycu.layout",  
 "schema": {...} // schema for rendering of the Backup Configuration options dialog, see section "Layout schema definition"  
 }  
}

### POST /api/resources - list sub-resources

*Stateless modules only.*

This endpoint is called during backup to compose a list of sub-resources to back up. Responses are used to build a catalog describing the backup.

Request body:

{  
 "ctx": {  
 // ctx returned by the list call on the parent resource  
 },  
 "authInfo": {  
 ...  
 },  
 "excludes": [   
 // optional list of ctx from items excluded by a user during in the Backup Configuration dialog  
 // field is skipped if the module does not support /api/config/backup endpoint  
 {"ctx": {...}} // ctx of excluded item   
 ],  
 "options": {   
 // optional json object configured by a user in the Backup Configuration dialog  
 // the content is defined by a possibleOptions.schema field returned by /api/config/backup endpoint  
 // the field is skipped if the module does not support /api/config/restore endpoint  
 }  
 "paginationCtx": {   
 // optional; module-specific context for pagination from previous response.  
 // First request will always omit this field.  
 ...  
 }  
}

Response body:

{  
 "resources": [  
 {  
 "name": "string", // display name of the root resource  
 "type": "string" // type of the root resource  
 "canBackup": false, // boolean, if resource supports backup  
 "backupSeqGroup": 1, // integer, sequence group number for backup  
 "restoreSeqGroup": 1, // integer, sequence group number for restore  
 "hasSubresources": false, // boolean, true if resource can be expanded  
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 }  
 }  
 ],  
 "hasMoreRecords": true, // optional; boolean, true if request should be repeated to obtain more data  
 "paginationCtx": { // optional; module-specific context for pagination. Should be set if hasMoreRecords is set to true.  
 },  
 "information": [ // optional: informational output  
 "string",  
 "string",  
 ]  
  
}

* Response format is similar to /api/roots, but omits id and displayCtx.
* Response may contain an errors field (see “Error propagation”). Any errors returned are reported as warnings in the HYCU backup task report.
* Response may contain an information field. Field contents are reported as informational messages that do not alter the job status.
* The field hasMoreRecords indicates the response is incomplete and should be repeated. For details, see “Pagination”.
* The field paginationCtx lets the module provide the pagination state used for the next request. Should be set if hasMoreRecords is set to true.

### POST /api/backup - backup a (sub)resource

*Stateless modules only.*

This endpoint is called during backup to issue a backup of a (sub)resource.

Request body:

{  
 "taskUuid": "uuid", // unique ID of the backup set,  
 "targetPath": "string", // DEPRECATED. Fully qualified bucket URL where to put data. The field is absent if the module declares REMOTE\_STORAGE\_ONLY capability feature in /api/version response  
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 },  
 "authInfo": {  
 ...  
 },  
 "excludes": [  
 // optional, see description of similar field of /api/resources request   
 ],  
 "options": {  
 // optional, see description of similar field of /api/resources request   
 },  
 "storage": { // optional, storage backend authentication; see "Backup storage"  
 // This field is absent if,  
 // - the module declares REMOTE\_STORAGE\_ONLY capability.features in /api/version response, or  
 // - (legacy) capabilities.supportedStorageTypes is not set in /api/version response, and (deprecated) field targetPath is used  
 ...  
 }  
}

Response body:

{  
 "taskUuid": "uuid", // unique ID of the backup set given in the request  
 "workingPath": "string", // Deprecated: (STORAGE\_GC and STORAGE\_S3 only) path of backup artifacts; sub-directory of storage.url from request.   
 "status": "string", // enum, one of: PENDING, RUNNING, DONE, ERROR, UNKNOWN   
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 },  
 "information": [ // optional: informational output  
 "string",  
 "string",  
 ]  
}

* DEPRECATED: Field targetPath contains a fully qualified bucket URL.
  + Field targetPath field is deprecated by use of storage field.
* Field status may return PENDING or RUNNING to handle asynchronous SaaS API calls or multiphase backup operations.
* DEPRECTED (STORAGE\_GC and STORAGE\_S3 only) Field workingPath should be a fully qualified bucket URL starting with storage.url (or targetPath, if set).
* DEPRECATED: (STORAGE\_GC and STORAGE\_S3 only) Field workingPath should be the same for all responses pertaining to a taskUuid - (i.e. a “root” folder of a particular backup).
* Field workingPath should be absent if the module declares REMOTE\_STORAGE\_ONLY capability, or STORAGE\_SERVICE storage type.
* If response status is ERROR, its sub-resources will not be backed up. This applies to related /api/status calls.
* If response status is DONE, but response contained the errors field (see “Error propagation”), such errors are reported as warnings in the HYCU task report.
* Response may contain an information field. Field contents are reported as informational messages that do not alter the job status.

### POST /api/bulk/backup - backup entire backupable

*Stateful modules only*

This endpoint is called during backup to handle backup of entire root resource with all subresources, including building of the catalog.

Request body:

{  
 "taskUuid": "uuid", // unique ID of the backup set,  
 "catalogPath": "path/to/catalog", // path in module container where module should output the catalog  
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 },  
 "authInfo": {  
 ...  
 },  
 "excludes": [  
 // optional, see description of similar field of /api/resources request   
 ],  
 "options": {  
 // optional, see description of similar field of /api/resources request   
 },  
 "storage": { // optional, storage backend authentication; see "Backup storage"  
 // This field is absent if the module declares REMOTE\_STORAGE\_ONLY capability.features in /api/version response, or  
 ...  
 }  
}

Response body:

{  
 "taskUuid": "uuid", // unique ID of the backup set given in the request  
 "status": "string", // enum, one of: PENDING, RUNNING, DONE, ERROR, UNKNOWN   
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 }  
}

* Field status may return PENDING or RUNNING and track state via /api/status.
* After backup is complete, module must provide the catalog database in the path indicated by field catalogPath.
* Returned catalog must adhere to the schema described in section “Catalog schema”.

#### Catalog schema

For bulk operations, HYCU expects the catalog in sqlite3 format with the following schema:

-- table of resources, hierarchical  
  
CREATE TABLE `backup\_catalog\_resources` (  
 `id` INTEGER PRIMARY KEY AUTOINCREMENT ,   
  
 `name` VARCHAR ,   
  
 `type` VARCHAR ,   
  
-- enum string: FAILED, DONE  
 `backupStatus` VARCHAR ,  
  
 `parentId` INTEGER ,  
  
 `depth` INTEGER );  
  
-- index to accelerate parent membership queries  
CREATE INDEX `backup\_catalog\_resources\_parentId\_idx` ON `backup\_catalog\_resources` ( `parentId` );  
  
-- table to track types used to display scenarios  
CREATE TABLE `types` (`id` INTEGER PRIMARY KEY AUTOINCREMENT , `name` VARCHAR NOT NULL , UNIQUE (`name`));

Module must create the catalog database and make it available in path indicated by field catalogPath at the conclusion of the bulk backup. Module may add other fields in the schema to track its own state needed for restore. Catalog database should contain only what is needed to successfully restore any (sub)resource.

#### Error schema

For bulk operations, HYCU expects sthe catalog to contain the following table describing errors that occurred during bulk backup or restore:

-- table of errors for bulk operations  
  
CREATE TABLE `errors` (  
 `id` INTEGER PRIMARY KEY AUTOINCREMENT ,  
 `name` VARCHAR , /\* resource name or ID \*/   
 `type` VARCHAR, /\* resource type \*/  
 `severity` VARCHAR, /\* severity: ERROR, WARNING \*/  
 `message` VARCHAR , /\* error message \*/  
 `code` INTEGER, /\* error code \*/  
 `time` INTEGER /\* unix timestamp in seconds, UTC \*/  
);

This table is considered transient - HYCU will drop this table after processing.

### POST /api/scenarios/restore - list possible restore scenarios

This endpoint is called when a user selects a restore action on specific backup. The response is used for rendering the restore scenarios in the restore dialog in HYCU UI.

This endpoint is optional. If the module does not support the endpoint (returns 404 or 501 HTTP status), HYCU UI renders the possible scenarios based on the Backup Catalog types.

Request body:

{  
 "authInfo": {  
 ...  
 },  
 "rootCtx": {  
 // resource[].ctx returned as response of /api/roots call  
 },  
 "types": ["resourceType1", ...] // list of backed up resource types  
}

Response body:

{  
 "restoreScenarios": [  
 {  
 "type": "resourceType1" // resource type  
 "name": "Attachment", // scenarios name to be shown in restore dialog in HYCU UI  
 "description": "RestoreAttachment", //optional description to be shown in restore dialog in HYCU UI  
 },...  
 ]  
}

* restoreScenarios should contain one scenario per type.
* The module may skip the types that are not supported for restore.

### POST /api/config/restore - restore options

This endpoint is called when a user selects a restore of the specific resource of a backupable. The response is used for rendering the restore options in the “Restore” dialog in HYCU UI.

This endpoint is optional. If the module does not support the endpoint (returns 404 or 501 HTTP status), HYCU UI does not render any options in the “Restore” dialog.

Request body:

{  
 "authInfo": {  
 ...  
 },  
 "rootCtx": {  
 // resource[].ctx returned as response of /api/roots call  
 },  
 "type": "resourceType1" //resource type selected by user during Restore Configuration dialog  
}

Response body:

{  
 "possibleOptions": { //optional, if missing no options are shown in HYCU UI  
 "schemaType": "hycu.layout",  
 "schema": {...} // //schema for rendering of Restore Configuration options, see section "Layout schema definition"  
 }  
}

### POST /api/restore - restore a (sub)resource

*Stateless modules only.*

This endpoint is called during restore to issue to restore of a (sub)resource. Request is built from the catalog created that HYCU persists after backup, which contains context data, and restore response contexts returned by restore of parent resources.

Request body:

{  
 "taskUuid": "uuid", // unique ID of the restore set  
 "workingPath": "string", // DEPRECATED: working path where artifacts are stored for this backup. Absent if the a module stores backup data on a remote storage.  
 "sourceCtx": {  
 // key-value map, module-specific context that was returned by /api/resources during backup, or last /api/roots before backup  
 },  
 "backupCtx": {  
 // key-value map, module-specific context returned during backup in a response for /api/backup or /api/status when status is DONE  
 },  
 "parentCtx": {  
 // key-value map, module-specific context returned by restore of the parent  
 },  
 "authInfo": {  
 ...  
 },  
 // optional, resource type selected by a user on the first step of restore wizard.   
 // The field is set if the "options" field is available  
 "optionsType": "string",   
 "options": {  
 // optional JSON object configured by a user during Restore Configuration flow  
 // the content is defined by possibleOptions.schema returned by /api/config/restore endpoint  
 // the field is skipped if the module does not support /api/config/backup endpoint  
 },  
 "storage": { // optional, storage backend information; see "Backup storage"  
 // This field is absent if,  
 // - the module declares REMOTE\_STORAGE\_ONLY capability.features in /api/version response, or  
 // - (legacy) capabilities.supportedStorageTypes is not set in /api/version response  
 ...  
 }   
}

Response body:

{  
 "taskUuid":"uuid", // unique ID of the restore set, given in the request  
 "status": "string", // string, one of: PENDING, RUNNING, DONE, ERROR, UNKNOWN  
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 },  
 "information": [ // optional: informational output  
 "string",  
 "string",  
 ]  
}

* DEPRECATED: workingPath is deprecated by use of storage field.
* A field ctx may be sent to sub-resources (if any) of the restored resource using parentCtx.
* If response status is ERROR, its sub-resources will not be restored. This applies to related /api/status calls.
* If response status is DONE, but response contained the errors field (see “Error propagation”), such errors are reported as warnings in the HYCU task report.
* Response may contain an information field. Field contents are reported as informational messages that do not alter the job status.

### POST /api/bulk/restore - restore entire backupable

*Stateful module only*

This endpoint is called during restore to issue to restore of a (sub)resource. Request is built from the catalog created by the module created upon the bulk backup completion. Items selected in the restore UI are sent in this request. Stateful module must expand this selection for example, restore a ticket may imply restore of its linked attachments.

Request body:

{  
 "taskUuid": "uuid", // unique ID of the restore set  
 "catalogPath": "path/to/catalog", // path in module container where module should read the catalog database  
 "authInfo": {  
 ...  
 },  
 // optional, resource type selected by a user on the first step of restore wizard.   
 // The field is set if the "options" field is available  
 "optionsType": "string",   
 "options": {  
 // optional JSON object configured by a user during Restore Configuration flow  
 // the content is defined by possibleOptions.schema returned by /api/config/restore endpoint  
 // the field is skipped if the module does not support /api/config/restore endpoint  
 },  
 "storage": { // optional, storage backend information; see "Backup storage"  
 // This field is absent if the module declares REMOTE\_STORAGE\_ONLY capability.features in /api/version response  
 ...  
 },  
 "selection": { // UI restore selection  
 "scenario": "type", // string, selected scenario to restore, as returned by /api/scenarios  
 "items": [ 41, 122, 12 ], // array of integers, primary key IDs of selected items in the catalog database  
 }  
}

Response body:

{  
 "taskUuid":"uuid", // unique ID of the restore set, given in the request  
 "status": "string", // string, one of: PENDING, RUNNING, DONE, ERROR, UNKNOWN  
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 }  
}

* Primary keys of selected items in the restore UI are sent in the field selection.items.
* Field status may return PENDING or RUNNING and track state via /api/status.
* Catalog database provided on path indicated by field catalogPath must adhere to the schema described in “Catalog schema”.

### POST /api/status - get status of backup/restore operation

This endpoint is called during backup and restore to allow the module to track status of asynchronous SaaS operations, or to advance state of a multiphase operation. This endpoint is called every 30 seconds until status indicates status to be DONE or ERROR.

Request body:

{  
 "taskUuid": "uuid", // unique ID of the backup or restore set  
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 },  
 "authInfo": {  
 ...  
 }  
 "storage": { // optional, storage backend information; see "Backup storage"  
 // This field is absent if,  
 // - the module declares REMOTE\_STORAGE\_ONLY capability.features in /api/version response, or  
 // - (legacy) capabilities.supportedStorageTypes is not set in /api/version response  
 ...  
 }   
  
}

The field ctx is passed from the last backup, restore or status call. The field storage is passed to allow multi-phase operations of the resource that may include storage backend interaction.

Response body:

{  
 "taskUuid": "uuid", // unique ID of the backup or restore set, given in the request  
 "status": "string" // enum, one of: PENDING, RUNNING, DONE, ERROR, UNKNOWN  
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 },  
 "information": [ // optional: informational output  
 "string",  
 "string",  
 ]  
  
}

* A field ctx from response is passed as ctx in a request for the next status call, allowing state transfer.
* A field ctx may be sent to sub-resources (if any) of the restored resource using parentCtx.
* Response may contain an information field. Field contents are reported as informational messages that do not alter the job status.

### POST /api/bulk/status - get status of bulk backup/restore operation

This endpoint is called during bulk backup and restore to allow the module to track status of a bulk operation. This endpoint is called every 30 seconds until status indicates status to be DONE or ERROR.

Request body:

{  
 "taskUuid": "uuid", // unique ID of the backup or restore set  
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 },  
 "authInfo": {  
 ...  
 }  
 "storage": { // optional, storage backend information; see "Backup storage"  
 // This field is absent if,  
 // - the module declares REMOTE\_STORAGE\_ONLY capability.features in /api/version response, or  
 // - (legacy) capabilities.supportedStorageTypes is not set in /api/version response  
 ...  
 }   
  
}

The field ctx is passed from the last backup, restore or status call. The field storage is passed to allow multi-phase operations of the resource that may include storage backend interaction.

Response body:

{  
 "taskUuid": "uuid", // unique ID of the backup or restore set, given in the request  
 "status": "string" // enum, one of: PENDING, RUNNING, DONE, ERROR, UNKNOWN  
 "progress": [ // optional, indicate progress  
 {  
 "id": "metric id", // string, metric identifier to track progress state  
 "text": "description", // string, human-readable metric description for display purposes  
 "value": 10000 // integer, value of  
 "max": 110000 // optional, integer, maximum expected value, used to infer percentage of completion from 'value' field  
 "unit": "kB" // optional, string, unit description  
 },  
 ...  
 ],  
 "summary": { // optional, summary of items processed. Applicable only for status DONE or ERROR.   
 "errors": { // count of resources that reported ERROR  
 "resource.type1": 120,  
 "resource.type2": 10  
 }  
 "warnings": { // count of resources that reported WARNING  
 "resource.type1": 100,  
 "resource.type": 11  
 },  
 "ok": { // count of resources that were processed without errors or warnings  
 "resource.type1": 10000,  
 "resource.type2": 100  
 }  
 },  
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 }  
}

* A field ctx from response is passed as ctx in a request for the next status call, allowing state transfer.
* A field ctx may be sent to sub-resources (if any) of the restored resource using parentCtx.
* Field progress can be used to indicate progress. This is useful for stateful modules using bulk operations.
* Field summary can be used to produce details of the number of items processed for each type and severity, at completion of the bulk operation.

### POST /api/abort

*Stateful module only*

This endpoint is called to trigger an abort a running bulk backup or restore operation. This is an out-of-band operation. Completion of bulk operation after the abort is issued is still tracked via /api/status endpoint. Module must handle aborts gracefully, propagating the abort request to any downstream SaaS operations.

Request body:

{  
 "taskUuid": "uuid", // uuid of the backup/restore task to abort  
}

Response body:

{}

### POST /api/retention/cleanup - remote storage cleanup

This endpoint is called during retention cleanup when a backup expires according to a policy and the backup is marked as a remote (workingPath field of an /api/backup response was absent). The endpoint is called once per a backup task.

Request body:

{  
 "taskUuid": "uuid", // unique ID of the cleanup task  
 "rootCtx": {  
 // resource[].ctx returned as response of /api/roots call  
 },  
 "backupCtx": {  
 // key-value map, module-specific context returned during a backup operation as a /api/backup or /api/status response when status is DONE  
 }  
 "authInfo": {  
 ...  
 },  
}

Response body:

{  
 "taskUuid": "uuid", // unique ID of the cleanup task  
 "status": "string" // enum, one of: PENDING, RUNNING, DONE, ERROR, UNKNOWN  
 "ctx": {  
 // key-value map, module-specific context - opaque to HYCU  
 }  
}

If response status is PENDING or RUNNING, the /api/status endpoint is called.

## Storage Service HTTP API reference

Storage service presents a module-local gateway to the backend storage for the given operation. Service is accessible via HTTP. URL for the service is sent via storage.url field to /api/backup, /api/restore and /api/status requests. Example:

{  
 ...  
 "storage": {  
 "url": "http://host.docker.internal:8081",  
 "type": "STORAGE\_SERVICE",  
 "authInfo": {}  
 }  
 ...  
}

Module can use the Storage Service to upload and download objects. Object names can contain any sequence of valid Unicode characters, when UTF-8 encoded, and must not contain Carriage Return or Line Feed characters.

### Error handling

On 4xx or 5xx response, Storage Service API returns the following object:

{  
 "error": {  
 "reason": "<string>", // reason, short code for error reason  
 "message": "<string>" // description of the error condition   
 },  
 "code": 404 // HTTP status code  
}

Example error response:  
  
```typescript  
{  
 ...  
 "error": {  
 "reason": "NOT\_FOUND",  
 "message": "Object 'd2035b78-e356-420a-b5ad-5cae0c626213/rsrc-1/data-dump.txt' does not exist in catalog."  
 },  
 "code": 404  
}

### POST /api/v2/objects/{objectPath} - upload object

Uploads object to the path specified with objectPath path parameter.

Successful upload returns the following response:

{  
 "kind":"POST <objectPath>", //  
 "id":"objects.post",  
 "items":  
 [  
 true  
 ],  
 "code":200  
}

Example response:

{  
 "kind":"POST 'objects/rsrc-1/data-dump.txt",  
 "id":"objects.post",  
 "items":  
 [  
 true  
 ],  
 "code":200  
}

Errors:

* *400 Bad Request:*
  + The request was malformed or missing required parameters.
  + Same object already exists.
* *500 Internal Server Error* - An unexpected error occurred on the server.

### GET /api/v2/objects/{objectPath} - download object

Returns a data stream of specified object by object path.

* Ranged requests are not supported.
* Response is using HTTP chunked encoding.

Errors:

* *400 Bad Request:* - The request was malformed or missing required parameters.
* *404 Not found:* - Object not found.
* *500 Internal Server Error* - An unexpected error occurred on the server.

## Language bindings (SDK)

HYCU provides a module SDK for Python 3.10+, including a library of common types, and a sample module with predefined dependencies for serving HTTP (flask, gunicorn). Module SDKs are documented separately.

Additional language bindings may be added later.

## Layout schema definition

The layout schema consists of two parts:

* layout - specifies the form layout, controllers and its properties and is formed of an array of elements.
* payload - specifies the payload that is sent on save. The value of the key is the ID of the controller that you wish the value to be fetched from.

### Layout positioning elements

These are basic building blocks for the positioning of controls:

Column:

{  
 "type": "column" // Creates a empty flex column block  
 "properties": {  
 "label": "Section label" // Add if you want a section label for grouping purposes  
 },  
 "elements":[] // Nested elements  
}

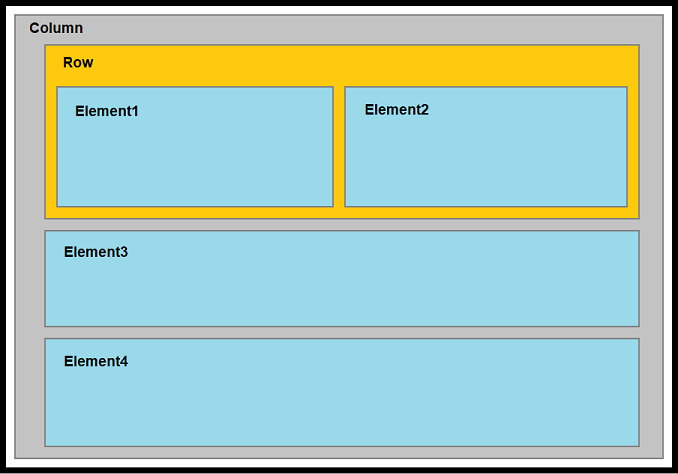
Row:

{  
 "type": "row", // Creates a empty flex row block  
 "elements":[] // Nested elements  
}

Example layout:

"layout": {  
 "elements": [  
 {  
 "type": 'column';  
 "elements": [  
 {  
 "type": 'row';  
 "elements": [{...element1}, {...element2}];  
 },  
 {  
 ...element3  
 },  
 {  
 ...element4  
 }   
 ];  
 }  
 ];  
 };

The output would look like this:



Displayed layout using the sample schema

### Layout control elements

Layout control elements have several types:

{  
 "id": 'exportFormat', // ID of the control that you also bind to then in the payload spec.  
 "type": 'dropdown', // type of the control  
 "value": '', // optional default value of the control based on control type  
 "properties": {  
 // properties specific for each control  
 },  
 "validators": {  
 // validators bound on control  
 },  
},  
  
Supported control element types and their default value types:  
  
 - text - string // Example: 'someString'  
  
 - dropdown - string[] // Example: ['elementValue1'] for single select or ['elementValue1', 'elementValue2'] for multiselect. Value has to match the value part of the dropdown item.  
  
 - switch - boolean // Example: true  
  
 - checkbox // Default value works a bit different and is set in properties  
  
 - date - string | string[] | number | number[] // Example: there are two options here for single pick or range pick. If you want to use unixtimestamp use 1676553210000 or [1676553210000, 1676553210000] for range. You can also use ISO format '2011-12-03T10:15:30+01:00' or ['2011-12-03T10:15:30+01:00', '2011-12-03T10:15:30+01:00'] for range.  
  
 - file-upload  
  
 - text-area - string // Example: 'someString'  
  
 - tag-input - string[] // Example: ['someString1', someString2, ...]  
   
 - key-value manager input

Supported properties per type:

Text input:

{  
 "inputType": string, // input type (text, password, number... default is text)  
 "label": string, // Top input label  
 "placeholder": string, // Placeholder inside of input  
 "iconTooltip": string, // Input label help icon tooltip text  
 "optional": boolean, // Mark as optional, sets optional label adjacent to input label  
 "hasClipboard": boolean, // Adds clipboard inside of input for content copy  
 "readonly": boolean, // Sets input as read only  
 "maxNumber": number, // Max number for number input type  
 "minNumber": number, // Max number for number input type  
}

Dropdown:

{  
 "label": string, // Top input label  
 "items": [ // Array of dropdown items  
 {  
 "label": string, // Item label seen in dropdown  
 "value": any, // Whatever value is sent in post(string, number, object...)  
 },  
 ...  
 ],  
 "multiselect": boolean, // Dropdown has multi selectable items  
 "searchable": boolean, // Dropdown items are searchable  
 "iconTooltip": string, // Input label help icon tooltip text  
 "optional": boolean, // Mark as optional, sets optional label adjacent to input label  
}

Switch:

{  
 "label": string, // Top input label  
 "subLabel": string, // Label put on right of switch button  
 "iconTooltip": string, // Input label help icon tooltip text  
}

Checkbox:

{  
 "label": string, // Top input label  
 "items": [ // Array of checkbox items, if you want only one checkbox put only one item  
 {  
 "label": string, // Item label  
 "value": any, // Whatever value is sent in post(string, number, object...)  
 "isSelected": boolean // Default value selection  
 },  
 ...  
 ],  
 "iconTooltip": string, // Input label help icon tooltip text  
}

Date time calendar:

{  
 "label": string, // Top input label  
 "iconTooltip": string, // Input label help icon tooltip text  
 "optional": boolean, // Mark as optional, sets optional label adjacent to input label  
 "pickerType": 'both' | 'calendar' | 'timer', // Type of date time picker  
 "selectMode": 'single' | 'range' | 'rangeFrom' | 'rangeTo', // Date picker selection mode  
 "showSeconds": boolean, // Show seconds in picker  
 "placeholder": string, // Placeholder inside of input  
}

File upload:

{  
 "label": string, // Top input label  
 "iconTooltip": string, // Input label help icon tooltip text  
 "optional": boolean, // Mark as optional, sets optional label adjacent to input label  
 "placeholder": string, // Placeholder inside of input  
 "fileTypes": string // Allowed file types ('\*.\*', \*.pdf, ...)  
}

Text area:

{  
 "label": string, // Top input label  
 "iconTooltip": string, // Input label help icon tooltip text  
 "optional": boolean, // Mark as optional, sets optional label adjacent to input label  
 "placeholder": string, // Placeholder inside of input  
 "hasClipboard": boolean, // Adds clipboard inside of input for content copy  
}

Tag input:

{  
 "label": string, // Top input label  
 "iconTooltip": string, // Input label help icon tooltip text  
 "optional": boolean, // Mark as optional, sets optional label adjacent to input label  
}

Key-Value Manager input:

{  
 "label": string, // Top input label  
 "iconTooltip": string, // Input label help icon tooltip text  
 "optional": boolean, // Mark as optional, sets optional label adjacent to input label  
 "value": { key:string; value:string; isDisabled?:boolean }[], //List of key value pairs if some predefined  
 "componentLabels": { // Component parts labels  
 "key": string, // Label of key input  
 "value": string, // Label of value input  
 "tableHeader": string, // Label of table header  
 },  
 "singleInput": boolean, // set to true if you want component to have single input entries (key only)  
 "allowNullValue": boolean, // set to true if null values are possible for each key  
 "maxItems": number, // set limit for maximum entries, default is unlimited  
 "hideValues": boolean // set to true if you want values to be hidden in table  
}

Supported validators:

{  
 "required": boolean, // Value required validator  
 "minMaxTextLength": [min,max], // Text length validator  
 "minMaxTextNumber": [min,max], // Number size validator  
 "pattern": { // Regex validator  
 "regex": Regex, // Regex to validate, example: /^[0-9]+$/  
 "message": string // Error message   
 },  
 "email": boolean,  
}

### Layout helper elements

Message box is used to position groups of validation messages in a dialog:

{  
 "type": 'message-box',   
 "properties": {  
 severity: 'string' // INFO, WARNING, ERROR  
 text: 'string'  
 }  
},

### Rules:

You can specify rules for controls and helper elements. Rules tell how controls behave under certain conditions. Lets say we have an example with a dropdown and a input:

{  
 "elements": [  
 {  
 "id": "dropdown1",  
 "type": "dropdown",  
 "value": [  
 "elementOne"  
 ],  
 "properties": {  
 "label": "Dropdown one",  
 "items": [  
 {  
 "label": "Element One",  
 "value": "elementOne"  
 },  
 {  
 "label": "Element Two",  
 "value": "elementTwo"  
 },  
 {  
 "label": "Element Three",  
 "value": "elementThree"  
 }  
 ],  
 "searchable": true  
 },  
 "validators": {  
 "required": true  
 }  
 },  
 {  
 "id": "username",  
 "type": "text",  
 "value": "someUsername",  
 "properties": {  
 "label": "Username",  
 "placeholder": "Test placeholder",  
 "iconTooltip": "Some input tooltip..."  
 },  
 "validators": {  
 "required": true  
 },  
 "rules": [  
 {  
 "effect": "HIDE",  
 "condition": {  
 "elementId": "dropdown1",  
 "elementValue": [  
 "elementOne"  
 ]  
 }  
 }  
 ]  
 }  
 ]  
}

We can have an array of rules for each control. The rules consist of:

* **“effect”** which tells what rule needs to trigger. Currently we support “**HIDE**”, “**DISABLE**”, “**WARNING**” “**UPDATE\_PROPERTY**” and “**UPDATE\_VALUE**” and
* **“condition”** which tells the condition of the rule that you want to trigger. The condition consists of the elementId and its elementValue. So in the top example the rule will be active when dropdown1 has elementOne picked.
* You can also say “not” to negate conditions:

"elementValue": {"not": ["elementOne"] }

* **“text”** applies only for WARNING effect and tells what text is specified for the warning block.

{  
 ...   
 "rules": [  
 {  
 "effect": "WARNING",  
 "text": "Some Warning Text...",  
 "condition": {  
 "elementId": "dropdown1",  
 "elementValue": [  
 "elementOne"  
 ]  
 }  
 }  
 ]  
}

* **“propertyName”** and “**propertyValue”** apply only for UPDATE\_PROPERTY effect and tell what property of the control you want to change and what value you want to apply for it. In the bottom example, the dropdown2 control will have its items changed based od dropdown1 value.

{  
 "id": "dropdown2",  
 "type": "dropdown",  
 "value": [  
 "elementOne"  
 ],  
 "properties": {  
 "label": "Dropdown two",  
 "items": [],  
 "searchable": true  
 },  
 "validators": {  
 "required": true  
 },  
 "rules": [  
 {  
 "effect": "UPDATE\_PROPERTY",  
 "condition": {  
 "elementId": "dropdown1",  
 "elementValue": [  
 "elementOne"  
 ]  
 },  
 "propertyName": "items",  
 "propertyValue": [  
 {  
 "label": "Element One",  
 "value": "elementOne"  
 },  
 {  
 "label": "Element Two",  
 "value": "elementTwo"  
 },  
 {  
 "label": "Element Three",  
 "value": "elementThree"  
 }  
 ]  
 },  
 {  
 "effect": "UPDATE\_PROPERTY",  
 "condition": {  
 "elementId": "dropdown1",  
 "elementValue": [  
 "elementTwo"  
 ]  
 },  
 "propertyName": "items",  
 "propertyValue": [  
 {  
 "label": "One",  
 "value": "one"  
 },  
 {  
 "label": "Two",  
 "value": "two"  
 },  
 {  
 "label": "Three",  
 "value": "three"  
 }  
 ]  
 },  
 {  
 "effect": "UPDATE\_PROPERTY",  
 "condition": {  
 "elementId": "dropdown1",  
 "elementValue": [  
 "elementThree"  
 ]  
 },  
 "propertyName": "items",  
 "propertyValue": [  
 {  
 "label": "111",  
 "value": "111"  
 },  
 {  
 "label": "222",  
 "value": "222"  
 },  
 {  
 "label": "333",  
 "value": "333"  
 }  
 ]  
 }  
 ]  
}

For UPDATE\_VALUE, the **elementValue** outside of the condition is the value that will patch the original value of the control that the rule is applied on.

Example:

{  
 "effect": "UPDATE\_VALUE",  
 "condition": {  
 "elementId": "dropdown1",  
 "elementValue": [  
 "elementThree"  
 ]  
 },  
 "elementValue": [  
 "222"  
 ]  
}

In this example, if dropdown1 will have elementThree picked, the value of the control that the rule is applied on will be patched to 222.

### Payload:

You can specify your own needed payload. To pass the control value to the specific key, write its ID in the value for that key.

"payload": {  
 "someGroup": {  
 "someKey1": 'control1',  
 "someKey2": 'control2',  
 "someKey3": 'control3',  
 },  
 "anotherGroup": {  
 "someKey1": 'control4',  
 }  
}

If there is a hidden control bound in the payload, that property will be removed from the payload.

### Full JSON Schema example:

{  
 "layout": {  
 "elements": [  
 {  
 "type": "column",  
 "elements": [  
 {  
 "type": "row",  
 "elements": [  
 {  
 "id": "username",  
 "type": "text",  
 "properties": {  
 "label": "Username",  
 "placeholder": "Test placeholder",  
 "tooltip": "Some input tooltip..."  
 },  
 "validators": {  
 "required": true  
 }  
 },  
 {  
 "id": "password",  
 "type": "text",  
 "properties": {  
 "label": "Password",  
 "placeholder": "Test placeholder",  
 "inputType": "password"  
 },  
 "validators": {  
 "required": true  
 }  
 }  
 ]  
 },  
 {  
 "id": "dropdown1",  
 "type": "dropdown",  
 "properties": {  
 "label": "Dropdown one",  
 "items": [  
 {  
 "label": "Ena",  
 "value": "ena"  
 },  
 {  
 "label": "Dva",  
 "value": "dva"  
 },  
 {  
 "label": "Tri",  
 "value": "tri"  
 }  
 ],  
 "multiselect": true,  
 "searchable": true  
 },  
 "validators": {  
 "required": true  
 }  
 }  
 ]  
 }  
 ]  
 },  
 "payload": {  
 "auth": {  
 "username": "username",  
 "password": "password"  
 },  
 "dropdown\_values": {  
 "drop1": "dropdown1"  
 }  
 }  
}

Gives such a result:



Dialog generated from sample layout schema