**Version 1.3**

MiX Integrate

Integration Guidelines

# Consuming data using the MiX Integrate REST API

MiX Integrate is the generally recommended option for integrating with the Unity platform, and is the starting point for most integrations.

While the API can be accessed using any REST-capable client software or framework, customers using .NET can take advantage of the publicly-available MiX.Integrate.Api.Client package for simplified programmatic interaction with the API. For the full list of endpoints and methods available, refer to the online REST API documentation.

While every customer's needs are unique, a number of operations are commonly required by many client systems:

* Retrieve / refresh master data.
* Live tracking.
* Retrieving historical data.
* Streaming of data.

To assist current and future integrators, we have compiled some recommendations for the above operations which include:

* the REST API endpoint to use.
* the equivalent API client package method.
* a brief description of the data returned.
* recommended call frequency.

## Common Operation 1: Retrieving and refreshing master data

This operation is common to most client applications. "Master data" is reference data (entities commonly referenced by their 64-bit MiX Fleet Manager identifiers) not subject to frequently changes - lists of assets, drivers, user-defined locations, etc. - and therefore does not need to be retrieved frequently. It should be retrieved and cached locally, then periodically refreshed.

REST: GET /api/organisationgroups

API Client: GroupsClient.GetAvailableOrganisations

Retrieves a list of available organisations:

* On-Road IoT (long) identifier: GroupId.
* Organisations are groups of type OrganisationGroup.
* **Refresh no more than once per day.**

### Groups

REST: GET /api/organisationgroups/subgroups/{organisationId}

API Client: GroupsClient.GetSubGroupsAsync(organisationId)

Gets an organisation’s group hierarchy as a tree of groups and subgroups:

* MiX Fleet Manager (long) identifier: GroupId.
* Cache locally, use to resolve group IDs.
* **Refresh periodically (max 1 or 2 times per day).**

### Assets

REST: GET /api/assets/group/{organisationId}

API Client: AssetsClient.GetAllAsync(organisationId)

Gets a list of all assets in an organisation:

* MiX Fleet Manager (long) identifier: AssetId.
* Includes legacy 16-bit FM identifier: FmVehicleId.
* Cache locally and use to resolve asset IDs.
* **Refresh periodically (max 4 – 6 times per day).**

### Drivers

REST: GET /api/drivers/organisation/{organisationId}

API Client: DriversClient.GetAllDriversAsync(organisationId)

Gets a list of all drivers in an organisation:

* MiX Fleet Manager (long) identifier: DriverId.
* Includes legacy 16-bit FM identifier FmDriverId.
* Cache locally and use to resolve driver IDs.
* **Refresh periodically (max 4 – 6 times per day).**

### Library Events

REST: GET /api/libraryevents/organisation/{organisationId}

API Client: DriversClient.GetAllAsync(LibraryEvents(organisationId)

Gets a list of an organisation’s defined events:

* MiX Fleet Manager (long) identifier: EventTypeId.
* Cache locally and use to resolve event type ids.
* **Refresh periodically (max 1 – 2 times per day).**

### Locations

REST: GET /api/locations/group/{organisationId}

API Client: LocationsClient.GetAllAsync(organisationId)

Gets all the user-defined locations of an organisation, optionally includes the shape definition:

* MiX Fleet Manager (long) identifier: LocationId.
* (Optionally) includes WKT shape definitions.
* Cache locally, use to resolve locations IDs, map-related activity.
* **Refresh periodically (max 1 or 2 times per day).**

## Common Operation 2: Live tracking

This operation is commonly used by client applications needing the **current** position of assets at any given time, usually (though not necessarily) for real-time monitoring or tracking purposes. The client application calls the API at regular intervals to get the up-to-date information on the position of each asset.

Few assets are in operation 24 hours a day which means that, between one request by the client application for the latest positions and the next, the back-end may not have received position updates for many of the assets associated with the requests. The data for such assets in the response to both calls will be identical, an unnecessary waste of both client and server resources.

To eliminate this needless repetition of data, client applications should **always** make use of the **cachedSince** parameter (the value of which should represent the UTC time of the previous call) to ensure that the response to a “GetLatest” call excludes assets for which no new data was received.

### Latest position - all assets in an organisation

REST: POST /api/positions/groups/latest/1  
 BODY: [organisationId]

API Client: PositionsClient.GetLatestByGroupIds(organisationId, 1)

Retrieves a list of the latest position of every asset in an organisation

* Calling this method with an **organisationId** is **strongly** recommended! It is the **fastest** and most efficient way to retrieve current positions, with the least amount of overhead.
* Calling this method with anything except an **organisationId** results in poor performance and is **strongly** discouraged .
* After calling this method, a client application must wait **at least 30 seconds** before calling the method again.

### Latest position - specific subset of assets in an organisation

REST: POST /api/positions/assets/latest/1  
 BODY: [assetId1, assetId2 [,...n]]

API Client: PositionsClient.GetLatestByAssetIds(List<assetId>, 1)

Retrieves the latest position of each asset in a specified list:

* Call this method when you only want positions for some of the assets in an organisation.
* When the list has more than 100-150 assetIds, you will usually get **faster** results by calling the method above to (retrieve positions for all assets in the organisation), and discarding the positions you don't need.
* After calling this method, a client application must wait **at least 30 seconds** before calling the method again.

### The delays mentioned here are of critical importance - failure to observe appropriate delays may lead to requests being rejected and/or access being blocked.

## Common Operation 3: Retrieving historical data using date-range endpoints

This operation is commonly used by client applications to retrieve data (trips / events / positions) associated with selected assets or drivers over a given period of time. For example:

* Retrieve trips for a date range for assets for fuel-consumption reporting
* Retrieve positions for plotting an asset's route on a map
* Retrieve over-speeding events for driver behaviour analysis

Customer needs are many and varied, and the REST API methods for accessing historical data are intended to be used for specific, targeted queries.

**Use of the REST API for retrieval of historical data in bulk is strictly prohibited. Customers seeking access to historical data in bulk (to populate a data warehouse, for example), should log a Support Request and request a data extract instead of pulling data from the API.   
NB: date-range endpoints should NEVER be used by an incremental / ongoing / periodic data retrieval strategy.**

# Retrieving historical trips

### Trips - for specific assets and a given date range

REST: POST /api/trips/assets/from/{from}/to/{to}?includeSubtrips=<true|false>

BODY: [assetId1, assetId2 [,...n]]

API Client: TripsClient.GetRangeForAssets(List<id>, {from}, {to},includeSubtrips)

Retrieves trips for the listed assets over the given date/time range:

* Date and times must be specified in UTC, max range = 7 days.
* Subtrip details are excluded by default, unless includeSubtrips = **true**.

### Trips - for specific drivers and a given date range

REST: POST /api/trips/drivers/from/{from}/to/{to}?includeSubtrips=<true|false>

BODY: [driverId1, driverId2 [,...n]]

API Client: TripsClient.GetRangeForDrivers(List<id>, {from}, {to},includeSubtrips)

Retrieves trips for the listed drivers over the given date/time range:

* Date and times must be specified in UTC, max range = 7 days.
* Subtrip details are excluded by default, unless includeSubtrips = **true**.

### Trips - for groups (not recommended except at org level)

REST: POST /api/trips/groups/from/{from}/to/{to}/entitytype/Asset?includeSubtrips=<true|false>

BODY: [organisationId]]

API Client: TripsClient.GetRangeForGroups({organisationId},{from},{to},"Asset",{includeSubtrips})

Retrieves trips in the organisation for the given date/time range:

* Date and times must be specified in UTC, max range = 7 days.
* Subtrip details are excluded by default, unless includeSubtrips = **true**.
* For performance reasons, this method should **ONLY** be called with an **organisationId**(so the EntityType is immaterial)

# Retrieving historical events

### Events - for specific assets and a given date range

REST: POST /api/events/assets/from/{from}/to/{to}

BODY: {

"EntityIds": [assetId1, assetId2 [,...n]],

"EventTypeIds": [eventTypeId1, eventTypeId2, [,..n]]

}

API Client: EventsClient.GetRangeForAssets({assetIds},{from},{to},{eventTypeIds})

Retrieves events for the listed assets over the given date/time range:

* Date and times must be specified in UTC, max range 7 days.
* Filter the results by EventTypeIds, leave empty or null to retrieve all events.

### Events - for specific drivers and a given date range

REST: POST /api/events/drivers/from/{from}/to/{to}

BODY: {

"EntityIds": [driverId1, driverId2 [,...n]],

"EventTypeIds": [eventTypeId1, eventTypeId2, [,..n]]

}

API Client: EventsClient.GetRangeForDrivers({driverIds}, {from},{to},{eventTypeIds})

Retrieves events for the listed drivers over the given date/time range:

* Date and times must be specified in UTC, max range 7 days.
* Filter the results by EventTypeIds, leave empty or null to retrieve all events

### Events - for groups (not recommended except at org level)

REST: POST /api/events/groups/from/{from}/to/{to}/entitytype/Asset

BODY: [organisationId]]

API Client: EventsClient.GetRangeForGroups({orgId},{from},{to},"Asset",{eventTypeIds}

Retrieves trips in the organisation for the given date/time range:

* Date and times must be specified in UTC, max range 7 days.
* Subtrips are excluded by default, unless includeSubtrips = **true**
* For performance reasons, this method should **ONLY** be called with an **organisationId** (so the EntityType is immaterial).

# Retrieving historical positions

### Positions - for specific assets and a given date range

REST: POST /api/positions/assets/from/{from}/to/{to}

BODY: [assetId1, assetId2 [,...n]]

API Client: PositionsClient.GetRangeForAssets({assetIds},{from},{to})

Retrieves positions for the listed assets over the given date/time range:

* Date and times must be specified in UTC, max range 7 days.

### Positions - for specific drivers and a given date range

REST: POST /api/positions/drivers/from/{from}/to/{to}

BODY: [driverId1, driverId2 [,...n]]

API Client: PositionsClient.GetRangeForDrivers({driverIds},{from},{to})

Retrieves positions for the listed drivers over the given date/time range:

* Date and times must be specified in UTC, max range 7 days.

### Positions - for groups (not recommended except at org level)

REST: POST /api/positions/groups/from/{from}/to/{to}/entitytype/Asset

BODY: [organisationId]]

API Client: PositionsClient.GetRangeForGroups({orgId},{from},{to},"Asset")

Retrieves positions for all assets in the organisation for the given date/time range:

* Date and times must be specified in UTC, max range 7 days.
* For performance reasons, this method should **ONLY** be called with an **organisationId** (so the EntityType is immaterial).

## Common Operation 4: Data Streaming

This operation is commonly used by client applications that need to retrieve data (trips / events / positions) from the MiX platform based **on the time at which data is added to the MiX back-end data store.** Data is not necessarily received by the MiX platform in chronological order (consider the deferred nature of passive data, for example), so this data retrieval operation is typically used as a way to adding newly-received data of any age to a client-side repository such as a data warehouse.  
  
(In the legacy FM Web Services API, this type of incremental update was facilitated by the use of record ID "bookmarks" and calls to **Get*xxx*SinceID** endpoints.)  
With MiX Integrate, clients retrieve newly-added data through calls to "CreatedSince" endpoints using timestamp-like tokens named **SinceTokens**.

Calls to CreatedSince endpoints:

* return data in the order in which it is RECEIVED by the back-end
* limit the number of records returned by each call
* may include more or fewer than the requested number of records in the response
* Include a header named **GetSinceToken** in the response data:
  + The client application **must** save the value of this header for use as the input SinceToken on the next call to the endpoint
* Include a header named **HasMoreItems** in the response data to indicate whether more data may be available or not:
  + **false**,indicates that all available data has been returned; this corresponds to HTTP response code 200 - the client **must** wait at least 30 seconds before calling the endpoint again
  + **true** indicates that the more data may be available; this corresponds to HTTP response code 206 - the client application should call the endpoint without delay

**Client applications that use CreateSince calls to retrieve data must, after making a CreateSince call, base the decision on whether or not to implement a delay before calling again solely on the value of the HasMoreItems response header or HTTP response code**

# Streaming of events

### Events - created since a given timestamp for specific assets

REST: POST /api/events/assets/createdsince/sincetoken/{sinceToken}/quantity/{quantity}

BODY: [assetId1, assetId2 [,...n]]

API Client: EventsClient.GetCreatedSinceForAssets({assetIds},{sinceToken},{quantity})

Retrieves events created since a specific timestamp for the specified assets:

* **sinceToken** may not be older than 7 days
* The **HasMoreItems** property or header of the response indicates whether more data after the retrieved quantity is available
* The **GetSinceToken** property or header of the response indicates the sinceToken to be used to retrieve subsequent data
* A client may call this method repeatedly to retrieve all available data, using the GetSinceToken returned by one call as the SinceToken for the next call, **as long as HasMoreItems is TRUE**
* When the **HasMoreItems** method or property of the response is **FALSE**, (indicating that all available data has been retrieved), the client must wait **a minimum of 30 seconds** before calling the method again to request new data

### Events - created since a given timestamp for specific drivers

REST: POST /api/events/drivers/createdsince/sincetoken/{sinceToken}/quantity/{quantity}

BODY: [driverId1, driverId2 [,...n]]

API Client: EventsClient.GetCreatedSinceForDrivers({driverIds},{sinceToken},{quantity})

Retrieves events created since a specific timestamp for the specified drivers:

* **sinceToken** may not be older than 7 days
* The **HasMoreItems** property or header of the response indicates whether more data after the retrieved quantity is available
* The **GetSinceToken** property or header of the response indicates the sinceToken to be used to retrieve subsequent data
* A client may call this method repeatedly to retrieve all available data, using the GetSinceToken returned by one call as the SinceToken for the next call **as long as HasMoreItems is TRUE**
* When the **HasMoreItems** method or property of the response is **FALSE**, (indicating that all available data has been retrieved), the client must wait **a minimum of 30 seconds** before calling the method again to request new data

### Events – created since for organisation

REST: GET /api/events/groups/createdsince/organisation/{orgId}/sincetoken/{sinceToken}/quantity/{quantity}

API Client: EventsClient.GetCreatedSinceForOrganisation({orgId},{sinceToken},{quantity})

Retrieves events created since a specific timestamp for an organisation:

* **sinceToken** may not be older than 7 days
* The **HasMoreItems** property or header of the response indicates whether more data after the retrieved quantity is available
* The **GetSinceToken** property or header of the response indicates the sinceToken to be used to retrieve subsequent data
* A client may call this method repeatedly to retrieve all available data, using the GetSinceToken returned by one call as the SinceToken for the next call **as long as HasMoreItems is TRUE**
* When the **HasMoreItems** method or property of the response is **FALSE**, (indicating that all available data has been retrieved), the client must wait **a minimum of 30 seconds** before calling the method again to request new data

### Events – created since for organisation filtered by EventTypeId

REST: POST /api/events/groups/createdsince/organisation/{orgId}/sincetoken/{sinceToken}/quantity/{quantity}  
BODY: [eventTypeId1, eventTypeId2 [,.. n]]

API Client: EventsClient.GetCreatedSinceForOrganisationFiltered({orgId},{sinceToken},{quantity},List<eventTypeId>)

Retrieves events created since a specific timestamp for an organisation, filtered by EventTypeId:

* **sinceToken** may not be older than 7 days
* The **HasMoreItems** property or header of the response indicates whether more data after the retrieved quantity is available
* The **GetSinceToken** property or header of the response indicates the sinceToken to be used to retrieve subsequent data
* A client may call this method repeatedly to retrieve all available data, using the GetSinceToken returned by one call as the SinceToken for the next call **as long as HasMoreItems is TRUE**
* When the **HasMoreItems** method or property of the response is **FALSE**, (indicating that all available data has been retrieved), the client must wait **a minimum of 30 seconds** before calling the method again to request new data

# Streaming of positions

### Positions – created since for entities in groups since a specified timestamp (not recommended except at site level)

REST: POST /api/positions/groups/createdsince/entitytype/Asset/sincetoken/{sinceToken}/quantity/{quantity} BODY: [siteId1, siteId2 [,...n]]

API Client: PositionsClient.GetCreatedSinceForGroups({siteIds},"Asset",{sinceToken},{quantity})

Retrieves positions created since a specific timestamp for entities in the specified groups:

* **sinceToken** may not be older than 7 days
* EntityTypecan be Asset or Driver
* The **HasMoreItems** property or header of the response indicates whether more data after the retrieved quantity is available
* The **GetSinceToken** property or header of the response indicates the sinceToken to be used to retrieve subsequent data
* A client may call this method repeatedly to retrieve all available data, using the GetSinceToken returned by one call as the SinceToken for the next call **as long as HasMoreItems is TRUE**
* When the **HasMoreItems** method or property of the response is **FALSE**, (indicating that all available data has been retrieved), the client must wait **a minimum of 30 seconds** before calling the method again to request new data

### Positions – created since for organisation

REST: GET /api/positions/groups/createdsince/organisation/{orgId}/sincetoken/{sinceToken}/quantity/{quantity}

API Client: PositionsClient.GetCreatedSinceForOrganisation({orgId},{sinceToken},{quantity})

Retrieves positions for an organisation created since a specific timestamp (sinceToken):

* **sinceToken** may not be older than 7 days
* The **HasMoreItems** property or header of the response indicates whether more data after the retrieved quantity is available
* The **GetSinceToken** property or header of the response indicates the sinceToken to be used to retrieve subsequent data
* A client may call this method repeatedly to retrieve all available data, using the GetSinceToken returned by one call as the SinceToken for the next call **as long as HasMoreItems is TRUE**
* When the **HasMoreItems** method or property of the response is **FALSE**, (indicating that all available data has been retrieved), the client must wait **a minimum of 30 seconds** before calling the method again to request new data

# Streaming of trips

### Trips - created since a given timestamp for specific assets

REST: POST /api/trips/assets/createdsince/sincetoken/{sinceToken}/quantity/{quantity}

?includeSubtrips=<true|false>

BODY: [assetId1, assetId2 [,...n]]

API Client: TripsClient.GetCreatedSinceForAssets({assetIds},{sinceToken},{quantity},includeSubtrips)

Retrieves trips created since a specific timestamp for the specified assets

* **sinceToken** may not be older than 7 days
* The **HasMoreItems** property or header of the response indicates whether more data after the retrieved quantity is available
* The **GetSinceToken** property or header of the response indicates the sinceToken to be used to retrieve subsequent data
* A client may call this method repeatedly to retrieve all available data, using the GetSinceToken returned by one call as the SinceToken for the next call **as long as HasMoreItems is TRUE**
* When the **HasMoreItems** method or property of the response is **FALSE**, (indicating that all available data has been retrieved), the client must wait **a minimum of 30 seconds** before calling the method again to request new data

### Trips - created since a given timestamp for specific drivers

REST: POST /api/trips/drivers/createdsince/sincetoken/{sinceToken}/quantity/{quantity}

?includeSubtrips=<true|false>

BODY: [driverId1, driverId2 [,...n]]

API Client: TripsClient.GetCreatedSinceForDrivers({driverIds},{sinceToken},{quantity},includeSubtrips)

Retrieves trips created since a specific timestamp for the specified drivers

* **sinceToken** may not be older than 7 days
* The **HasMoreItems** property or header of the response indicates whether more data after the retrieved quantity is available
* The **GetSinceToken** property or header of the response indicates the sinceToken to be used to retrieve subsequent data
* A client may call this method repeatedly to retrieve all available data, using the GetSinceToken returned by one call as the SinceToken for the next call **as long as HasMoreItems is TRUE**
* When the **HasMoreItems** method or property of the response is **FALSE**, (indicating that all available data has been retrieved), the client must wait **a minimum of 30 seconds** before calling the method again to request new data

### Trips – created since a given timestamp for organisation

REST: GET /api/trips/groups/createdsince/organisation/{organisationId}  
/sincetoken/{sinceToken}/quantity/{quantity}?includeSubtrips=<true|false>

API Client: TripsClient.GetCreatedSinceForOrganisation({organisationId},{sinceToken},{quantity}, includeSubtrips)

Retrieves trips created since a specific timestamp for an organisation

* **sinceToken** may not be older than 7 days
* The **HasMoreItems** property or header of the response indicates whether more data after the retrieved quantity is available
* The **GetSinceToken** property or header of the response indicates the sinceToken to be used to retrieve subsequent data
* A client may call this method repeatedly to retrieve all available data, using the GetSinceToken returned by one call as the SinceToken for the next call **as long as HasMoreItems is TRUE**
* When the **HasMoreItems** method or property of the response is **FALSE**, (indicating that all available data has been retrieved), the client must wait **a minimum of 30 seconds** before calling the method again to request new data

# Consuming data using a data feed

There are instances where, after unit configurations have been streamlined and data bloat eliminated, the sheer volume of data involved becomes burdensome, even for an efficient and well-implemented client solution, and an alternative way of consuming data generated by units is necessary.

In such instances, the transfer of certain types of data via MiX Integrate's "pull"-based mechanism can be replaced by a "push"-oriented system: a MiX data feed.

**NB:**

* Every data feed incurs daily operating costs and significant maintenance and management overhead.
* A data feed should never be used as a panacea for a poorly-designed integration or badly-configured fleet

With a MiX data feed, any (or all) of the following types of data can be pushed to a target destination (owned and operated by the consumer) as such data is received and processed, alleviating the consumer system of the burden of retrieving the data from MiX Integrate:

* Trips
* Events
* Positions

Note: data feeds do not negate the need for MiX Integrate. Master data, for example, will still need to be retrieved and refreshed periodically.

Once configured and activated, a data feed monitors the data store for trips, events, and/or positions belonging to the organisation(s) which it is configured to service. As each qualifying record is added to the data store, the data feed posts it as a message in MiX Integrate JSON format to the target destination, where it can be consumed by the client application.

Currently, data feeds support posting to:

* AWS Kinesis Data Streams (a separate stream per data type)
* Microsoft Azure Event Hubs (a separate entity path per data type)
* REST API WebHooks

In all cases, the target destination(s) belong to the consumer. The consumer remains wholly responsible for the operation, maintenance, management, and security of the destination(s), and wholly liable for all costs arising from the operation and/or use thereof by any party, including Powerfleet.