morango Documentation

Learning Equality

Mar 18, 2024

CONTENTS

1 Overview **3**
- 1.1 Motivating user story . . . 3
- 1.2 Objectives . . . 4
- 1.3 Usage in Kolibri . . . 4

2 Architecture **5**
- 2.1 Profiles . . . 5
- 2.2 Syncable models . . . 5
- 2.3 Partitions . . . 6
- 2.4 Filters and scopes . . . 7
- 2.5 Certificates . . . 8
- 2.6 Session controller, contexts, and operations . . . 10

3 Syncing **13**
- 3.1 Concepts . . . 13
- 3.2 Process . . . 13
- 3.3 Orchestration . . . 14
- 3.4 Signals . . . 15

4 IDs and Counters **17**
- 4.1 Identifiers . . . 17
- 4.2 Counters . . . 17

5 Merging **19**
- 5.1 Fast-forward merges . . . 19
- 5.2 Merge conflicts . . . 20

6 Deletion **21**
- 6.1 Soft-deletion . . . 21
- 6.2 Hard-deletion . . . 21

7 API **23**
- 7.1 Models . . . 23
- 7.2 Sync sessions . . . 46
- 7.3 Viewsets . . . 50
- 7.4 Permissions . . . 52

Python Module Index **55**

Index **57**

Morango is a Django database replication engine written in pure Python. It is designed and maintained in support of the Kolibri product ecosystem.

CHAPTER

ONE

OVERVIEW

Morango is a pure-Python database replication engine for Django that supports peer-to-peer syncing of data. It is structured as a Django app that can be included in projects to make specific application models syncable.

Developed in support of the Kolibri product ecosystem, Morango includes some important features including:

- A certificate-based authentication system to protect privacy and integrity of data
- A change-tracking system to support calculation of differences between databases across low-bandwidth connections
- A set of constructs to support data partitioning

1.1 Motivating user story

Imagine a scenario where we have four instances of Kolibri:

- *Home* is a tablet used at home by a learner with no internet access
- *Facility* is a laptop at a nearby school, also with no internet access
- *City* is a laptop in a nearby city
- *Cloud* is a server online in the cloud

On *Facility*, a coach assigns resources to a learner's user account. The learner brings *Home* to the school and syncs with *Facility*, getting only their assignments and no private data about other learners.

The learner uses *Home* for a week, engaging with the assigned resources. They (and other learners) bring their tablets back to school and sync again with *Facility*. The coach can now see the recent user engagement data for their class.

An admin user wants to get the recent user engagement data from the *Facility* device onto their *City* device. In order to achieve this, the admin may bring *City* to the remote area. Once *City* arrives in the remote area, *Facility* and *City* can sync over the school's local network.

Finally, the admin brings *City* back to the city and syncs with *Cloud* over the internet. At this point, *Facility*, *City*, and *Cloud* all have the same data. Now, imagine a second admin in another city syncs their own laptop (*City 2*) with *Cloud*. Now they too would have the recent data from *Facility*.

1.2 Objectives

- **User experience:** Streamline the end-user syncing process as much as possible
- **Privacy:** Only sync data to devices and users authorized to access that data
- **Flexibility:** Afford the ability to sync only a subset of the data
- **Efficiency:** Minimize storage, bandwidth, and processing power
- **Integrity:** Protect data from accidental and malicious data corruption
- **Peer-to-peer:** Devices should be able to communicate without a central server
- **Eventual consistency:** Eventually all devices will converge to the same data

1.3 Usage in Kolibri

Morango is not the only way that Kolibri instances communicate with each other and other services. Some other ways Kolibri communicates are:

- Discovering other Kolibri instances with Zeroconf
- Calling REST APIs for getting meta-information about discovered Kolibri instances
- Calling REST APIs for sending anonymous usage statistics to an LE telemetry server
- Calling REST APIs for browsing content available for import from Studio and other Kolibri instances
- Downloading static channel database and media files from Studio and other Kolibri instances

Morango's certificate, change-tracking, and partitioning features are useful especially in situations where diff-based updates and guarantees about distributed data consistency and coherence are useful.

CHAPTER

TWO

ARCHITECTURE

2.1 Profiles

A **profile** is a unique, semantically meaningful name within the Kolibri ecosystem. It corresponds to a set of interrelated *syncable models* that "make sense" when synced together.

Currently there is just a single profile in the Kolibri ecosystem: `facilitydata`.

2.2 Syncable models

A **syncable model** is a Django model which can be synced between devices using Morango. Every syncable model is associated with exactly one *profile*, and exactly one *partition* within the profile.

To make a Django model syncable, inherit from `SyncableModel`. All subclasses need to define:

- `morango_profile` - the name of the model's profile
- `morango_model_name` - a unique name within the profile
- `calculate_source_id` - a method that returns a unique ID of the record
- `calculate_partition` - a method that returns the *partition string* of the record

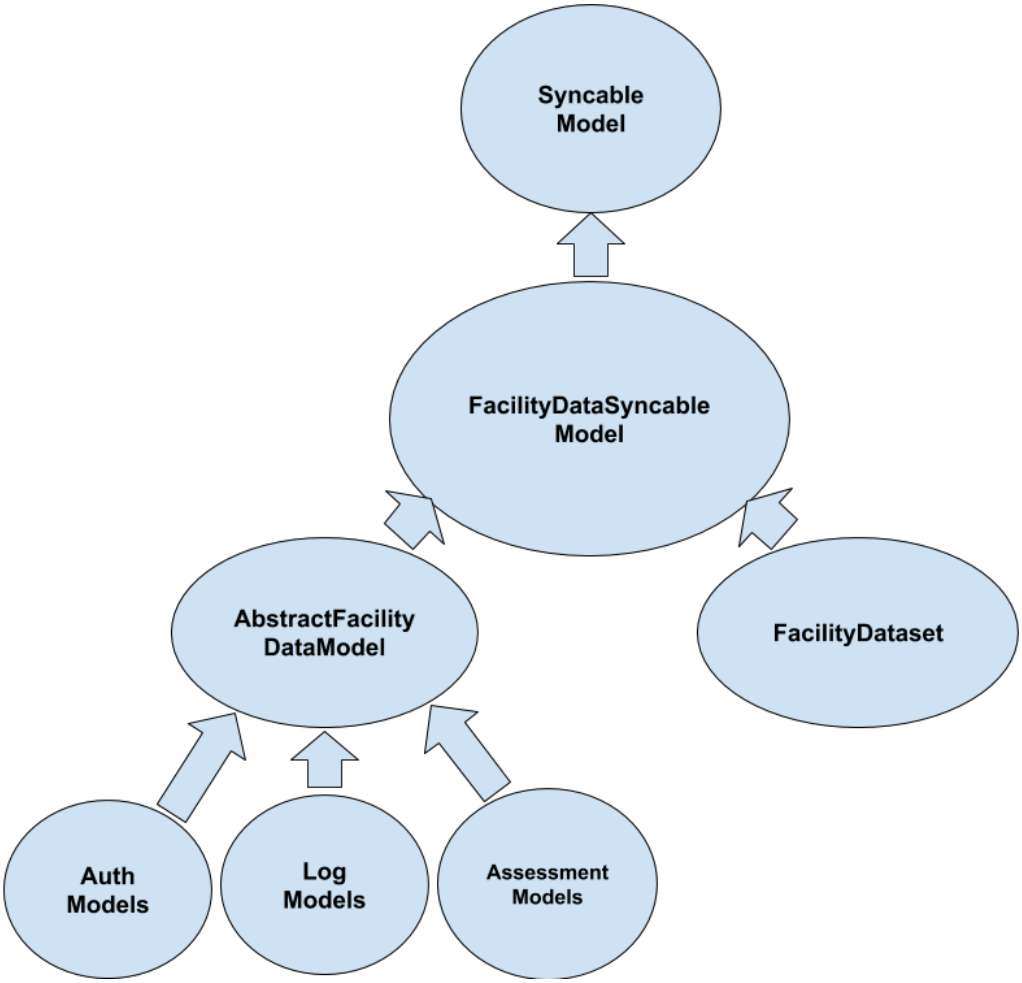
There are some constraints to Django models that are serialized and synced in Morango:

- models must not have self-referential foreign keys or dependency loops
- models must not use relationships based on Django generic foreign keys
- models must not use many-to-many relationships

In order to ensure that schema migrations work cleanly, always provide default values when defining model fields on syncable models.

If you create custom querysets or managers and your model inherits from `SyncableModel`, then your custom classes should also inherit from `SyncableModelQuerySet` or `SyncableModelManager` in order to maintain syncability for these models.

In Kolibri, we currently define a base `SyncableModel` called `FacilityDataSyncableModel`. Both `FacilityDataset` and `AbstractFacilityDataModel` inherit from this. In turn, other syncable Kolibri models inherit from `AbstractFacilityDataModel` as shown below:



2.3 Partitions

A **partition** is a string that defines a subset of the *syncable models* in a *profile*. Taken together, the partitions of a profile define mutually exclusive and complete segmented coverage of all syncable model records.

Partition strings use colon characters to delimit levels of a hierarchy, and Python template strings to dynamically insert source IDs of models. Aside from this, Morango places no constraints on the structure of partition strings, and they can be constructed using any convention or strategy. A leading part part of a colon-delimited partition string designating some parent partition is called a **partition prefix**.

As a hypothetical example, a record for a syncable model like a content interaction log might be associated with a syncable user in a syncable facility. The combination of the user ID and the facility ID could be used to dynamically define a partition like `${facility_id}:${user_id}` for that and other similar records. "Containment" of partitions in the hierarchy can be checked with a simple Python `startswith` string check between partitions. In the example above, the partition `${facility_id}:${user_id}` is said to be contained by the partition `${facility_id}` for user `U1` in facility `F1` because `"F1:U1".startswith("F1") == True` and `F1` is the partition prefix.

In Kolibri, we currently have five mutually-exclusive partitions in the `facilitydata` profile, where the source ID of the facility is the `dataset_id`:

- **everyone has write-only access**
  - partition string: `${dataset_id}:anonymous`

  - used for content session logs

- **all authenticated users have read-only access**
  - partition string: `${dataset_id}:allusers-ro`
  - used for facility metadata, classes, and other collections
- **a learner has personalized read-only access**
  - partition string: `${dataset_id}:user-ro:${user_id}`
  - used for user roles and membership in classes and groups
- **a learner has personalized read and write access**
  - partition string: `${dataset_id}:user-rw:${user_id}`
  - used for content interaction logs
- **everything else**
  - partition string: `${dataset_id}`
  - used for quizzes and lessons

Note that all facility models share the prefix `${dataset_id}`, which means that they are all "contained" in that top-level partition.

2.4 Filters and scopes

A **filter** is a set of *partition prefixes* represented as an end-line-delimited string. A **scope** is a set of filters which defines the permissions conferred by a *certificate* and stored in a `ScopeDefinition` object.

When designing scopes – i.e. composing scopes from filters and partitions – care must be taken to ensure that foreign keys in synced models refer to other models that were also synced in the same scope. Otherwise, an alternative would be to ensure that the application can gracefully handle missing records when necessary because there would be no guarantee of coherence.

As of this writing, there are currently two scope definitions defined in Kolibri for the `facilitydata` profile:

- The `full-facility` scope provides full read and write access to all data related to a facility. This includes the facility model itself plus associated classes, lessons, users, groups, content interaction logs, and everything else related to running a typical Kolibri classroom server.
- The `single-user` scope provides some of the access needed by a single learner, specifically the content interaction logs. Note that this does *not* currently include all necessary data. For example, lessons that have been assigned to the user are not in this scope, and must currently be synced through another mechanism (as yet to be determined).

Kolibri's scope definition fixture is shown below. Here, note that the `single-user` scope allows the user to write content-related logs and to read other facility data so that Kolibri is still able to function properly.

```
[
  {
    "model": "morango.scopedefinition",
    "pk": "full-facility",
    "fields": {
      "profile": "facilitydata",
      "version": 1,
      "primary_scope_param_key": "dataset_id",
```

(continues on next page)

(continued from previous page)

```
      "description": "Allows full syncing for data under the Facility with␣
↪FacilityDataset ID ${dataset_id}.",
      "read_filter_template": "",
      "write_filter_template": "",
      "read_write_filter_template": "${dataset_id}"
    }
  },
  {
    "model": "morango.scopedefinition",
    "pk": "single-user",
    "fields": {
      "profile": "facilitydata",
      "version": 1,
      "primary_scope_param_key": "",
      "description": "Allows syncing data for FacilityUser ${user_id} under Facility␣
↪with FacilityDataset ID ${dataset_id}.",
      "read_filter_template": "${dataset_id}:allusers-ro\n${dataset_id}:user-ro:${user_
↪id}",
      "write_filter_template": "${dataset_id}:anonymous",
      "read_write_filter_template": "${dataset_id}:user-rw:${user_id}"
    }
  }
]
```

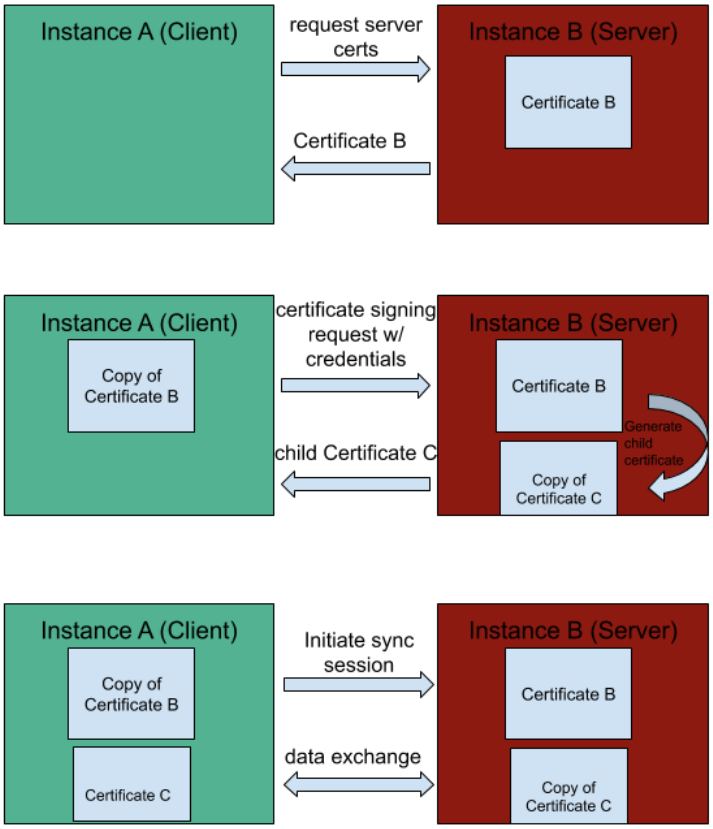
2.5 Certificates

Certificates are hierarchical pairs of private/public keys that grant device-level permission to sync data within a *filtered scope* of a *profile*. Once a device has been granted access to a scope of a profile, that device can grant that scope or a subset of it to other devices by generating child certificate pairs.

Scope access and the chain of trust are established as follows:

- The private key associated with a parent certificate can be used to issue a child certificate to another device with at most the permission granted by the scope of the parent certificate
- The child certificate can be used by the new device to allow it to prove to other devices that it is authorized to access the scope
- The entire chain of signed certificates back to the origin must be exchanged during sync between devices, and the signatures and hierarchy must be verified

In the example below, *Instance A* is able to establish a future sync relationship with *Instance B* by providing admin credentials to *Instance B* and requesting a signed certificate:

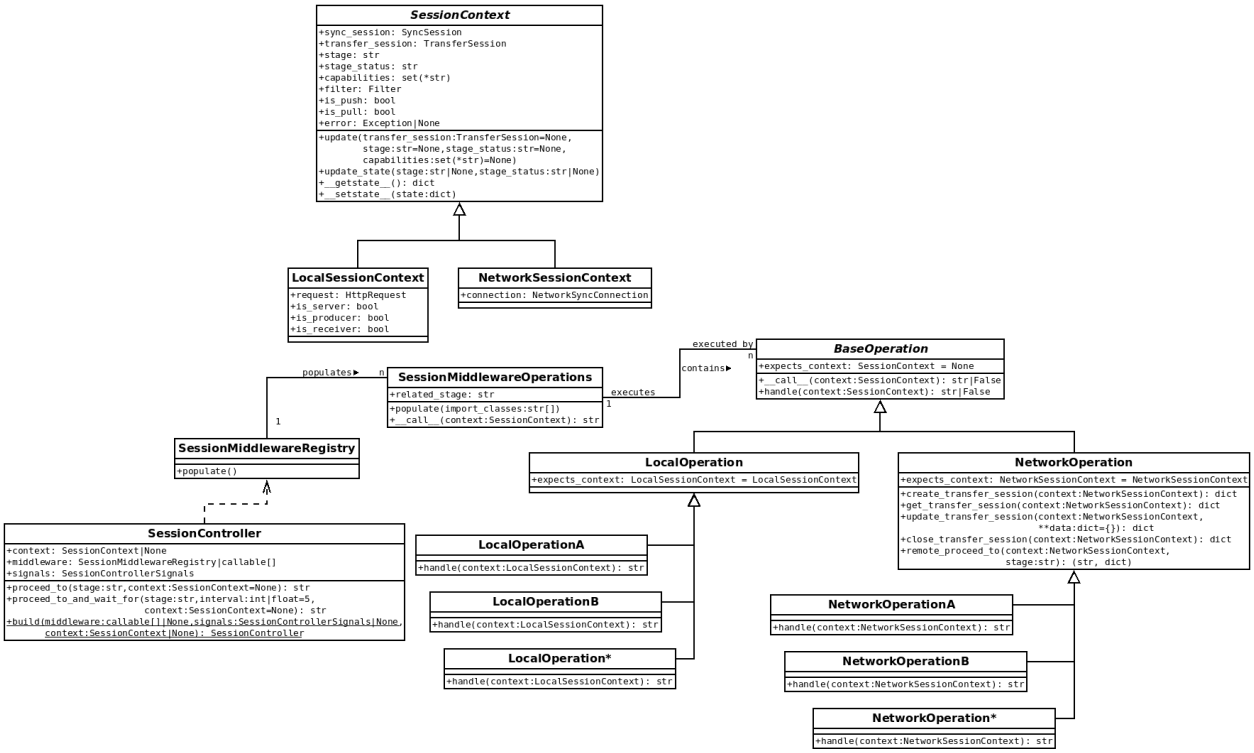


It should be cautioned that there is currently no mechanism for revoking certificates. This means that a stolen or hijacked device will have access to all data it has been granted, and updates to that data when another device is on the same network.

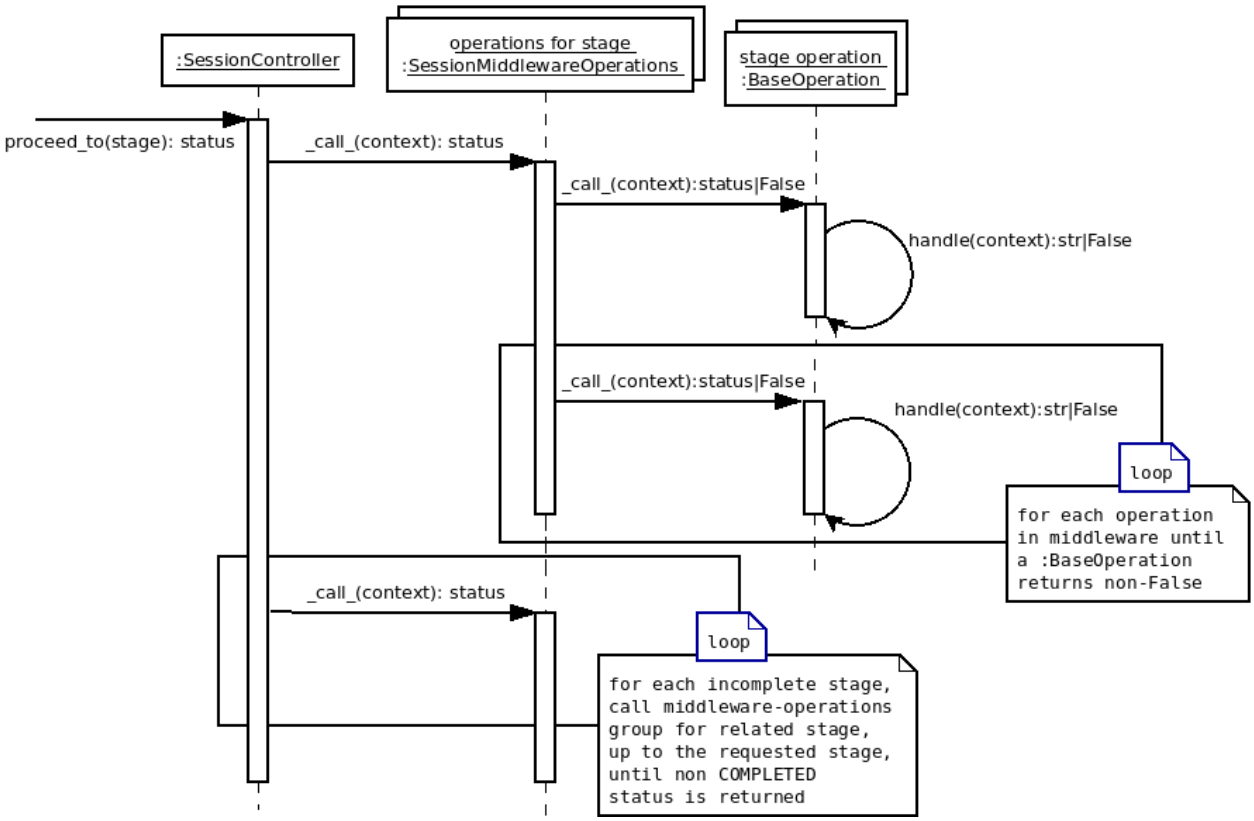
In Kolibri, on the `FacilityDataset` model, we generate the certificate as a function of the `calculate_source_id` method. Note that we currently set the ID of the certificate to be the same as the ID of the facility model. This allows queries on the certificate hierarchy tree to find certificates that are associated with the facility.

There's flexibility in the application layer for determining the validity of a root certificate, and it's specified on a per-profile basis. For the `facilitydata` profile, Kolibri leverages its `auth` models for this.

2.6 Session controller, contexts, and operations



A unidirectional sync has several stages: `INITIALIZING`, `SERIALIZING`, `QUEUING`, `TRANSFERRING`, `DEQUEUING`, `DESERIALIZING`, and `CLEANUP`. Each stage requires callable objects, referred to here simply as *operations*. Operations handle the necessary operational aspects of the transfer for each stage. The `SessionController` class establishes an internal API for invoking those operations through a Chain-of-responsibility software design pattern. Provided with a *context*, either a `LocalSessionContext` or a `NetworkSessionContext`, the controller will iterate through each incomplete stage and invoke the operations for stage, passing along the context object. An operation isn't required to handle the context, which is analogous to a request object, but can defer responsibility to the next operation in the stage's list of operations by returning `False`. At least one operation must handle the context, which is communicated by returning a `transfer_statuses` constant of either `PENDING`, `STARTED`, `ERRORED`, or `COMPLETED`.



The list of operations for each stage are configured through Django settings. The configuration key for each stage follows the pattern `MORANGO_%STAGE%_OPERATIONS`, so the list/tuple of operations for the `QUEUING` stage access the `MORANGO_QUEUING_OPERATIONS` configuration value. Built-in operations implement a callable `BaseOperation` class by overriding a `handle` method. The `BaseOperation` class supports raising an `AssertionError` to defer responsibility to the next operation.

CHAPTER

THREE

SYNCING

3.1 Concepts

The **store** holds serialized versions of syncable models. This includes both data that is on the current device and data synced from other devices.

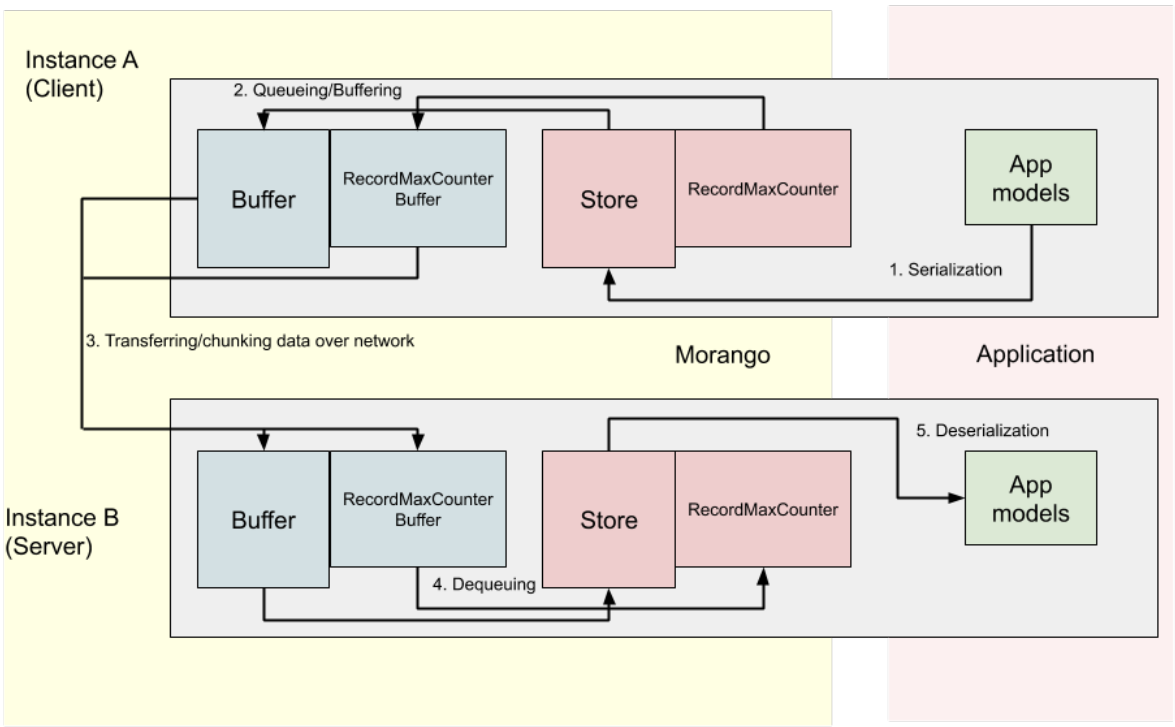
The **outgoing buffer** and **incoming buffer** mirror the schema of the store. They also include a transfer session ID which used to identify sets of data that are being synced as a coherent group to other Morango instances.

3.2 Process

Syncing is the actual exchange of data in a sync session. The general steps for syncing data are:

1. **Serialization** - serializing data that is associated with Django models in the Application layer, and storing it in JSON format in a record in the Store
2. **Queuing/Buffering** - storing serialized records and their modification history to a separate Buffers data structure
3. **Transfer/chunking of data** - the actual transfer of data over a request/response cycle in chunks of 500 records at a time
4. **Dequeuing** - merging the data received in the receiving buffers to the receiving store and record-max counter
5. **Deserialization** - merging data from the receiving Store into the Django models in the Application layer

In the illustration below, the application layer (on the right) is where app data resides as Django models, and the Morango layer (on the left) is where the Morango stores, counters, and buffers reside. *Instance A* (on the top) is sending data to *Instance B* (on the bottom). Application Django models in *Instance A* are serialized in JSON format and saved to the store. Data is queued in the buffers on *Instance A*, and then transmitted to the corresponding buffers on *Instance B*. The data is then integrated into the store and Django app models on *Instance B*.



3.3 Orchestration

In order to facilitate synchronization between several Morango instances, it can be convenient to create a Django management command which uses the Morango machinery.

For example, in Kolibri we have created a management command called kolibri manage sync. Note that any time this command is run, we always both pull and push, which guarantees that both Kolibri databases will have the same data afterwards.

Of particular importance is the `MorangoProfileController` which can create a `NetworkSyncConnection` with another Morango instance.

Once the client establishes a network connection, both instances must exchange certificates so that they can prove that they have the proper permissions in order to push or pull the data. If the client side lacks the proper certificates, they should use the network connection to do a `certificate_signing_request`, where they enter admin credentials of the other instance to generate a certificate with the valid permissions.

Once both sides have the proper certificates, the client can initiate a sync session with `create_sync_session`. This creates a `SyncClient` that can handle either pushing or pulling data to/from the other Morango instance.

3.4 Signals

During the sync process, Morango fires a few different signals from `signals` in `PullClient` and `PushClient`. These can be used to track the progress of the sync.

There are four signal groups:

- `session`
- `queuing`
- `transferring`
- `dequeuing`

Each signal group has 3 stages that can be fired:

- `started`
- `in_progress`
- `completed`

For a push or pull sync lifecycle, the order of the fired signals would be as follows:

1) Session started
2) Queuing started
3) Queueing completed
4) Transferring started
5) Transferring in progress
6) Transferring completed
7) Dequeuing started
8) Dequeuing completed
9) Session completed

CHAPTER

FOUR

IDS AND COUNTERS

4.1 Identifiers

There is generally one Morango instance for every Kolibri instance, and each of these are identified by a unique Morango **instance ID**. The instance ID is calculated as a function of a number of system properties, and will change when those properties change. Changes to the instance ID are not fatal, but stability is generally preferable.

The **database ID** identifies the actual database being used by a Morango instance. If a database has been backed up and restored or copied to a different Morango instance, a new database ID should be generated to help other Morango instances that may have already seen the previous state of the database.

Each syncable model instance within the database is identified by a unique **model source ID**. This is calculated randomly by default and takes the calculated partition and Morango model name into account. Models can also define their own behavior by overriding `calculate_source_id`.

4.2 Counters

A **counter** is a monotonically increasing version number. Comparing two counter values associated with the same object will show which one is newer.

Whenever a syncable model record is modified, a unique combination of the Morango instance ID and an incrementing counter version are assigned to the record. This combination specifies the record version.

Morango instances use **record-max counters** to keep track of the maximum version each record has been saved at. This is used to determine drive different merge behaviors during the sync process.

The **database-max counter** table tracks a mapping of scope filter strings to lists of (instance ID, counter) pairs. These (instance ID, counter) pairs reflect different Morango instances that have been previously synced at some counter value.

Morango sends **filter-max counters** to determine what data is already shared before syncing to efficiently determine the difference in data. Filter-max counters are the highest counters associated with every instance ID for both a filter and its supersets.

CHAPTER

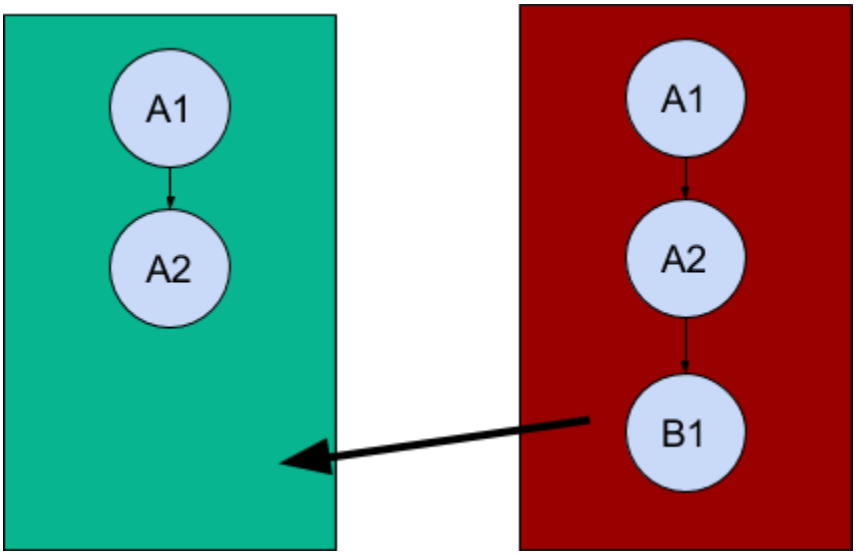
FIVE

MERGING

There are two possible cases for the merging of data: fast-forward merges and conflicts.

5.1 Fast-forward merges

A "fast-forward" data merge situation means that there is no conflict to resolve. This can be determined by checking if the current version of a record on the receiving device is already contained in the history of the transmitting device, or vice-versa.

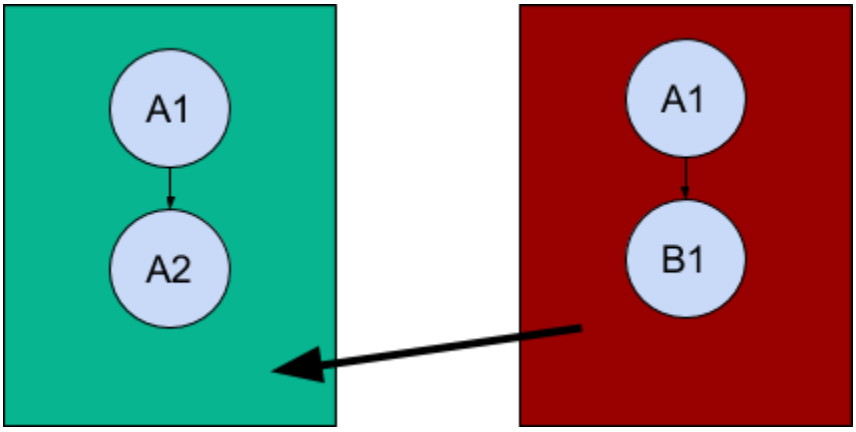


In the illustration above:

1. *Device A* (green) produces a new record, `r`. It gets assigned record version `A1` and history `[ A1 ]`.
2. Next, *Device A* modifies `r`. The record version changes to `A2` and the history is now `[ A2, A1 ]`.
3. *Device B* (red) now syncs data with *Device A* and both the devices have same version and history of record `r`.
4. *Device B* modifies its copy of `r` and sets the record version to `B1`. The history of `r` is now `[ B1, A2, A1 ]` on *Device B* and still `[ A2, A1 ]` on *Device A*.
5. When *Device A* syncs with *Device B* again (the arrow), there is no conflict and the update `B1` can be incorporated directly.

5.2 Merge conflicts

A merge conflict means that two devices have made changes to a record, and it is not clear how to reconcile the two change histories.



In the illustration above:

1. As above, *Device A* (green) produces a new record `r` with version `A1` and history `[ A1 ]`.
2. *Device B* (red) now syncs data with *Device A* and both the devices have same copy of record `r`.
3. Next, *Device B* modifies its copy of `r`. The record version changes to `B1` and the history `[ B1, A1 ]`.
4. *Device A* modifies its own copy of record `r` and saves it as `A2` with history `[ A2, A1 ]`.
5. When *Device A* syncs data with *Device B* again (the arrow), there is a conflict because both devices have modified `r`.

It is up to the implementing application to determine what the merge conflict resolution strategy is.

CHAPTER

SIX

DELETION

6.1 Soft-deletion

Typically, deletion merely hides records, rather than actually erasing data.

When a record for a subclass of `SyncableModel` is deleted, its ID is added to the `DeletedModels` table. When a subsequent serialization occurs, this information is used to turn on the `deleted` flag in the store for that record. When syncing with other Morango instances, the soft deletion will propagate to the store record of other instances.

This is considered a "soft-delete" in the store because the data is not actually cleared.

6.2 Hard-deletion

There are times, such as GDPR removal requests, when it's necessary to actually to erase data.

This is handled using a `HardDeletedModels` table. Subclasses of `SyncableModel` should override the `delete` method to take a `hard_delete` boolean, and add the record to the `HardDeletedModels` table when this is passed.

On serialization, Morango clears the `serialized` field entry in the store for records in `HardDeletedModels` and turns on the `hard_deleted` flag. Upon syncing with other Morango instances, the hard deletion will propagate to the store record of other instances.

CHAPTER

SEVEN

API

7.1 Models

class morango.models.**Buffer**(**args*, ***kwargs*)

Bases: `AbstractStore`

`Buffer` is where records from the internal store are queued up temporarily, before being sent to another morango instance, or stored while being received from another instance, before dequeuing into the local store.

Parameters

- **id** (*AutoField*) – Id
- **profile** (*CharField*) – Profile
- **serialized** (*TextField*) – Serialized
- **deleted** (*BooleanField*) – Deleted
- **hard_deleted** (*BooleanField*) – Hard deleted
- **last_saved_instance** (UUIDField) – Last saved instance
- **last_saved_counter** (*IntegerField*) – Last saved counter
- **partition** (*TextField*) – Partition
- **source_id** (*CharField*) – Source id
- **model_name** (*CharField*) – Model name
- **conflicting_serialized_data** (*TextField*) – Conflicting serialized data
- **_self_ref_fk** (*CharField*) – self ref fk
- **transfer_session_id** (ForeignKey to ~) – Transfer session
- **model_uuid** (UUIDField) – Model uuid

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

model_uuid

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

rmcb_list()

transfer_session

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

transfer_session_id

class morango.models.**Certificate**(*id*, *parent*, *profile*, *scope_definition*, *scope_version*, *scope_params*, *public_key*, *salt*, *serialized*, *signature*, *_private_key*)

Bases: `MPTTModel`, *UUIDModelMixin*

Parameters

- **id** (UUIDField) – Id
- **parent_id** (ForeignKey to ~) – Parent
- **profile** (*CharField*) – Profile
- **scope_definition_id** (ForeignKey to ~) – Scope definition
- **scope_version** (*IntegerField*) – Scope version
- **scope_params** (*TextField*) – Scope params
- **public_key** (PublicKeyField) – Public key
- **salt** (*CharField*) – Salt
- **serialized** (*TextField*) – Serialized
- **signature** (*TextField*) – Signature
- **_private_key** (PrivateKeyField) – private key
- **lft** (*PositiveIntegerField*) – Lft
- **rght** (*PositiveIntegerField*) – Rght
- **tree_id** (*PositiveIntegerField*) – Tree id
- **level** (*PositiveIntegerField*) – Level

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

certificate_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

check_certificate()

classmethod deserialize(*serialized*, *signature*)

classmethod generate_root_certificate(*scope_def_id*, ***extra_scope_params*)

get_scope()

has_private_key()

level

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

lft

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

parent

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

parent_id

property private_key

profile

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

public_key

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

rght

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

salt

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

classmethod save_certificate_chain(*cert_chain*, *expected_last_id=None*)

scope_definition

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

scope_definition_id

scope_params

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

scope_version

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

serialize()

serialized

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

sign(*value*)

sign_certificate(*cert_to_sign*)

signature

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

syncsessions_client

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

syncsessions_server

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

tree_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

uuid_input_fields = ('public_key', 'profile', 'salt')

verify(*value*, *signature*)

class morango.models.**DatabaseIDModel**(**args*, ***kwargs*)

Bases: *UUIDModelMixin*

Model to be used for tracking database ids.

Parameters

- **id** (UUIDField) – Id
- **current** (*BooleanField*) – Current
- **date_generated** (*DateTimeField*) – Date generated
- **initial_instance_id** (*CharField*) – Initial instance id

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

current

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

date_generated

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_next_by_date_generated(*, *field=<django.db.models.fields.DateTimeField: date_generated>*, *is_next=True*, ***kwargs*)

classmethod get_or_create_current_database_id()

get_previous_by_date_generated(*, *field=<django.db.models.fields.DateTimeField: date_generated>*, *is_next=False*, ***kwargs*)

initial_instance_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instanceidmodel_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <morango.models.core.DatabaseIDManager object>

save(**args*, ***kwargs*)

Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

uuid_input_fields = 'RANDOM'

class morango.models.**DatabaseMaxCounter**(**args*, ***kwargs*)

Bases: `AbstractCounter`

`DatabaseMaxCounter` is used to keep track of what data this database already has across all instances for a particular partition prefix. Whenever 2 morango instances sync with each other we keep track of those partition prefixes from the filters, as well as the maximum counter we received for each instance during the sync session.

Parameters

- **id** (*AutoField*) – Id
- **instance_id** (UUIDField) – Instance id
- **counter** (*IntegerField*) – Counter
- **partition** (*CharField*) – Partition

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

classmethod calculate_filter_specific_instance_counters(*filters*, *is_producer=False*, *v2_format=False*)

classmethod get_instance_counters_for_partitions(*partitions*, *is_producer=False*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

partition

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

classmethod update_fsics(*fsics*, *sync_filter*, *v2_format=False*)

class morango.models.**DeletedModels**(**args*, ***kwargs*)

Bases: Model

DeletedModels helps us keep track of models that are deleted prior to serialization.

Parameters

- **id** (UUIDField) – Id
- **profile** (*CharField*) – Profile

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

profile

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class morango.models.**Filter**(*template*, *params={}*)

Bases: object

contains_partition(*partition*)

is_subset_of(*other*)

class morango.models.**HardDeletedModels**(**args*, ***kwargs*)

Bases: Model

HardDeletedModels helps us keep track of models where all their data must be purged (*serialized* is nullified).

Parameters

- **id** (UUIDField) – Id
- **profile** (*CharField*) – Profile

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

profile

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class morango.models.**InstanceIDModel**(**args*, ***kwargs*)

Bases: Model

InstanceIDModel is used to track what the current ID of this Morango instance is based on system properties. If system properties change, the ID used to track the morango instance also changes. During serialization phase, we associate the current instance ID, as well as its counter with all the records that were serialized at the time.

Parameters

- **id** (UUIDField) – Id
- **platform** (*TextField*) – Platform
- **hostname** (*TextField*) – Hostname
- **sysversion** (*TextField*) – Sysversion
- **node_id** (*CharField*) – Node id
- **database_id** (ForeignKey to ~) – Database
- **counter** (*IntegerField*) – Counter
- **current** (*BooleanField*) – Current
- **db_path** (*CharField*) – Db path
- **system_id** (*CharField*) – System id

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

counter

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

current

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

database

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

database_id

db_path

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

classmethod get_current_instance_and_increment_counter()

classmethod get_or_create_current_instance(*clear_cache=False*)

Get the instance model corresponding to the current system, or create a new one if the system is new or its properties have changed (e.g. new MAC address).

get_proquint()

hostname

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property instance_info

Getter to access custom instance info defined in settings :return: dict

node_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

platform

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

system_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

sysversion

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

uuid_input_fields = ('platform', 'hostname', 'sysversion', 'node_id', 'database_id', 'db_path')

class morango.models.**Nonce**(**args*, ***kwargs*)

Bases: *UUIDModelMixin*

Stores temporary nonce values used for cryptographic handshakes during syncing. These nonces are requested by the client, and then generated and stored by the server. When the client then goes to initiate a sync session, it signs the nonce value using the private key from the certificate it is using for the session, to prove to the server that it owns the certificate. The server checks that the nonce exists and hasn't expired, and then deletes it.

Parameters

- **id** (*UUIDField*) – Id
- **timestamp** (*DateTimeField*) – Timestamp
- **ip** (*CharField*) – Ip

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

get_next_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=True*, ***kwargs*)

get_previous_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=False*, ***kwargs*)

ip

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

classmethod use_nonce(*nonce_value*)

uuid_input_fields = 'RANDOM'

class morango.models.**PrivateKeyField**(**args*, ***kwargs*)

Bases: RSAKeyBaseField

from_db_value(*value*, *expression*, *connection*)

get_prep_value(*value*)

Perform preliminary non-db specific value checks and conversions.

to_python(*value*)

Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

class morango.models.**PublicKeyField**(**args*, ***kwargs*)

Bases: RSAKeyBaseField

from_db_value(*value*, *expression*, *connection*)

get_prep_value(*value*)

Perform preliminary non-db specific value checks and conversions.

to_python(*value*)

Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

class morango.models.**RecordMaxCounter**(**args*, ***kwargs*)

Bases: AbstractCounter

RecordMaxCounter keeps track of the maximum counter each serialized record has been saved at, for each instance that has modified it. This is used to determine fast-forwards and merge conflicts during the sync process.

Parameters

- **id** (*AutoField*) – Id
- **instance_id** (UUIDField) – Instance id
- **counter** (*IntegerField*) – Counter
- **store_model_id** (ForeignKey to ~) – Store model

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

store_model

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

store_model_id

class morango.models.**RecordMaxCounterBuffer**(**args*, ***kwargs*)

Bases: AbstractCounter

RecordMaxCounterBuffer is where combinations of instance ID and counters (from RecordMaxCounter) are stored temporarily, until they are sent or received by another morango instance.

Parameters

- **id** (*AutoField*) – Id
- **instance_id** (UUIDField) – Instance id
- **counter** (*IntegerField*) – Counter
- **transfer_session_id** (ForeignKey to ~) – Transfer session
- **model_uuid** (UUIDField) – Model uuid

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

model_uuid

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

transfer_session

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

transfer_session_id

class morango.models.**Scope**(*definition*, *params*)

Bases: `object`

is_subset_of(*other*)

class morango.models.**ScopeDefinition**(*id*, *profile*, *version*, *primary_scope_param_key*, *description*, *read_filter_template*, *write_filter_template*, *read_write_filter_template*)

Bases: `Model`

Parameters

- **id** (*CharField*) – Id
- **profile** (*CharField*) – Profile
- **version** (*IntegerField*) – Version
- **primary_scope_param_key** (*CharField*) – Primary scope param key
- **description** (*TextField*) – Description
- **read_filter_template** (*TextField*) – Read filter template
- **write_filter_template** (*TextField*) – Write filter template
- **read_write_filter_template** (*TextField*) – Read write filter template

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

certificate_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

description

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_description(*params*)

get_scope(*params*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

primary_scope_param_key

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

profile

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

read_filter_template

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

read_write_filter_template

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

classmethod retrieve_by_id(*scope_def_id*)

version

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

write_filter_template

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class morango.models.**SharedKey**(**args*, ***kwargs*)

Bases: `Model`

The public key is publically available via the `api/morango/v1/publickey` endpoint. Applications who would like to allow certificates to be pushed to the server must also enable `ALLOW_CERTIFICATE_PUSHING`. Clients generate a `Certificate` object and set the `public_key` field to the shared public key of the server.

Parameters

- **id** (*AutoField*) – Id
- **public_key** (PublicKeyField) – Public key
- **private_key** (PrivateKeyField) – Private key
- **current** (*BooleanField*) – Current

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

current

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

classmethod get_or_create_shared_key(*force_new=False*)

Create a shared public/private key pair for certificate pushing, if the settings allow.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

private_key

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

public_key

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class morango.models.**Store**(**args*, ***kwargs*)

Bases: AbstractStore

Store is the concrete model where serialized data is persisted, along with metadata about counters and history.

Parameters

- **profile** (*CharField*) – Profile
- **serialized** (*TextField*) – Serialized
- **deleted** (*BooleanField*) – Deleted
- **hard_deleted** (*BooleanField*) – Hard deleted
- **last_saved_instance** (UUIDField) – Last saved instance
- **last_saved_counter** (*IntegerField*) – Last saved counter
- **partition** (*TextField*) – Partition
- **source_id** (*CharField*) – Source id
- **model_name** (*CharField*) – Model name
- **conflicting_serialized_data** (*TextField*) – Conflicting serialized data
- **_self_ref_fk** (*CharField*) – self ref fk
- **id** (UUIDField) – Id
- **dirty_bit** (*BooleanField*) – Dirty bit
- **deserialization_error** (*TextField*) – Deserialization error
- **last_transfer_session_id** (UUIDField) – Last transfer session id

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

deserialization_error

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

dirty_bit

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

last_transfer_session_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <morango.models.core.StoreManager object>

recordmaxcounter_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

class morango.models.**SyncSession**(**args*, ***kwargs*)

Bases: Model

SyncSession holds metadata for a sync session which keeps track of initial settings and the current transfer happening for this sync session.

Parameters

- **id** (UUIDField) – Id
- **start_timestamp** (*DateTimeField*) – Start timestamp
- **last_activity_timestamp** (*DateTimeField*) – Last activity timestamp
- **active** (*BooleanField*) – Active
- **is_server** (*BooleanField*) – Is server
- **client_certificate_id** (ForeignKey to ~) – Client certificate
- **server_certificate_id** (ForeignKey to ~) – Server certificate
- **profile** (*CharField*) – Profile
- **connection_kind** (*CharField*) – Connection kind
- **connection_path** (*CharField*) – Connection path
- **client_ip** (*CharField*) – Client ip

- **server_ip** (*CharField*) – Server ip
- **client_instance_id** (UUIDField) – Client instance id
- **client_instance_json** (*TextField*) – Client instance json
- **server_instance_id** (UUIDField) – Server instance id
- **server_instance_json** (*TextField*) – Server instance json
- **extra_fields** (*TextField*) – Extra fields
- **process_id** (*IntegerField*) – Process id

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

active

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

client_certificate

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

client_certificate_id

client_instance_data

client_instance_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

client_instance_json

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

client_ip

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

connection_kind

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

connection_path

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

extra_fields

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_connection_kind_display(*, *field=<django.db.models.fields.CharField: connection_kind>*)

get_next_by_last_activity_timestamp(*, *field=<django.db.models.fields.DateTimeField: last_activity_timestamp>*, *is_next=True*, ***kwargs*)

get_next_by_start_timestamp(*, *field=<django.db.models.fields.DateTimeField: start_timestamp>*, *is_next=True*, ***kwargs*)

get_previous_by_last_activity_timestamp(*, *field=<django.db.models.fields.DateTimeField: last_activity_timestamp>*, *is_next=False*, ***kwargs*)

get_previous_by_start_timestamp(*, *field=<django.db.models.fields.DateTimeField: start_timestamp>*, *is_next=False*, ***kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

is_server

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

last_activity_timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

process_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

profile

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

server_certificate

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

server_certificate_id

server_instance_data

server_instance_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

server_instance_json

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

server_ip

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

start_timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

transfersession_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class morango.models.**SyncableModel**(**args*, ***kwargs*)

Bases: *UUIDModelMixin*

`SyncableModel` is the base model class for syncing. Other models inherit from this class if they want to make their data syncable across devices.

Parameters

- **id** (`UUIDField`) – Id
- **_morango_dirty_bit** (*BooleanField*) – morango dirty bit
- **_morango_source_id** (*CharField*) – morango source id
- **_morango_partition** (*CharField*) – morango partition

ID_PLACEHOLDER = '${id}'

class Meta

Bases: `object`

abstract = False

cached_clean_fields(*fk_lookup_cache*)

Immediately validates all fields, but uses a cache for foreign key (FK) lookups to reduce repeated queries for many records with the same FK

Parameters

fk_lookup_cache – A dictionary to use as a cache to prevent querying the database if a FK exists in the cache, having already been validated

calculate_partition()

Should return a string specifying this model instance's partition, using *self.ID_PLACEHOLDER* in place of its own ID, if needed.

calculate_source_id()

Should return a string that uniquely defines the model instance or *None* for a random uuid.

calculate_uuid()

Should return a 32-digit hex string for a UUID that is calculated as a function of a set of fields from the model.

static compute_namespaced_id(*partition_value*, *source_id_value*, *model_name*)

deferred_clean_fields()

Calls *.clean_fields()* but excludes all foreign key fields and instead returns them as a dictionary for deferred batch processing

Returns

A dictionary containing lists of `ForeignKeyReference`s keyed by the name of the model being referenced by the FK

delete(*using=None*, *keep_parents=False*, *hard_delete=False*, **args*, ***kwargs*)

classmethod deserialize(*dict_model*)

Returns an unsaved class object based on the valid properties passed in.

classmethod merge_conflict(*current*, *push*)

morango_fields_not_to_serialize = ()

morango_model_dependencies = ()

morango_profile = None

objects

save(*update_dirty_bit_to=True*, **args*, ***kwargs*)

Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

serialize()

All concrete fields of the SyncableModel subclass, except for those specifically blacklisted, are returned in a dict.

class morango.models.**SyncableModelManager**(**args*, ***kwargs*)

Bases: ManagerFromSyncableModelQuerySet

class morango.models.**SyncableModelQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Bases: QuerySet

classmethod as_manager()

update(*update_dirty_bit_to=True*, ***kwargs*)

Update all elements in the current QuerySet, setting all the given fields to the appropriate values.

class morango.models.**TransferSession**(**args*, ***kwargs*)

Bases: Model

TransferSession holds metadata that is related to a specific transfer (push/pull) session between 2 morango instances.

Parameters

- **id** (UUIDField) – Id
- **filter** (*TextField*) – Filter
- **push** (*BooleanField*) – Push
- **active** (*BooleanField*) – Active
- **records_transferred** (*IntegerField*) – Records transferred
- **records_total** (*IntegerField*) – Records total
- **bytes_sent** (*BigIntegerField*) – Bytes sent
- **bytes_received** (*BigIntegerField*) – Bytes received
- **sync_session_id** (ForeignKey to ~) – Sync session
- **start_timestamp** (*DateTimeField*) – Start timestamp
- **last_activity_timestamp** (*DateTimeField*) – Last activity timestamp
- **client_fsic** (*TextField*) – Client fsic
- **server_fsic** (*TextField*) – Server fsic
- **transfer_stage** (*CharField*) – Transfer stage
- **transfer_stage_status** (*CharField*) – Transfer stage status

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

active

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

buffer_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

bytes_received

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

bytes_sent

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

client_fsic

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

delete_buffers()

Deletes *Buffer* and *RecordMaxCounterBuffer* model records by executing SQL directly against the database for better performance

filter

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_filter()

get_next_by_last_activity_timestamp(*, *field=<django.db.models.fields.DateTimeField: last_activity_timestamp>*, *is_next=True*, ***kwargs*)

get_next_by_start_timestamp(*, *field=<django.db.models.fields.DateTimeField: start_timestamp>*, *is_next=True*, ***kwargs*)

get_previous_by_last_activity_timestamp(*, *field=<django.db.models.fields.DateTimeField: last_activity_timestamp>*, *is_next=False*, ***kwargs*)

get_previous_by_start_timestamp(*, *field=<django.db.models.fields.DateTimeField: start_timestamp>*, *is_next=False*, ***kwargs*)

get_touched_record_ids_for_model(*model*)

get_transfer_stage_display(*, *field=<django.db.models.fields.CharField: transfer_stage>*)

get_transfer_stage_status_display(*, *field=<django.db.models.fields.CharField: transfer_stage_status>*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

last_activity_timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

property pull

Getter for *not push* condition, which adds complexity in conditional statements

push

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

recordmaxcounterbuffer_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

records_total

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

records_transferred

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

server_fsic

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

start_timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

sync_session

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

sync_session_id

transfer_stage

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

transfer_stage_status

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

update_state(*stage=None*, *stage_status=None*)

class morango.models.**UUIDField**(**args*, ***kwargs*)

Bases: `CharField`

Adaptation of Django's UUIDField, but with 32-char hex representation as Python representation rather than a UUID instance.

deconstruct()

Return enough information to recreate the field as a 4-tuple:

- The name of the field on the model, if contribute_to_class() has been run.
- The import path of the field, including the class:e.g. django.db.models.IntegerField This should be the most portable version, so less specific may be better.
- A list of positional arguments.
- A dict of keyword arguments.

Note that the positional or keyword arguments must contain values of the following types (including inner values of collection types):

- None, bool, str, int, float, complex, set, frozenset, list, tuple, dict
- UUID
- datetime.datetime (naive), datetime.date
- top-level classes, top-level functions - will be referenced by their full import path
- Storage instances - these have their own deconstruct() method

This is because the values here must be serialized into a text format (possibly new Python code, possibly JSON) and these are the only types with encoding handlers defined.

There's no need to return the exact way the field was instantiated this time, just ensure that the resulting field is the same - prefer keyword arguments over positional ones, and omit parameters with their default values.

from_db_value(*value*, *expression*, *connection*)

get_db_prep_value(*value*, *connection*, *prepared=False*)

Return field's value prepared for interacting with the database backend.

Used by the default implementations of get_db_prep_save().

get_default()

Returns the default value for this field.

get_internal_type()

prepare_value(*value*)

to_python(*value*)

Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

class morango.models.**UUIDModelMixin**(**args*, ***kwargs*)

Bases: Model

Mixin for Django models that makes the primary key "id" into a UUID, which is calculated as a function of jointly unique parameters on the model, to ensure consistency across instances.

Parameters
id (UUIDField) – Id

class Meta

Bases: object

abstract = False

calculate_uuid()

Should return a 32-digit hex string for a UUID that is calculated as a function of a set of fields from the model.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

save(**args*, ***kwargs*)

Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

uuid_input_fields = None

7.2 Sync sessions

class morango.sync.session.**SessionWrapper**

Bases: Session

Wrapper around *requests.sessions.Session* in order to implement logging around all request errors.

bytes_received = 0

bytes_sent = 0

prepare_request(*request*)

Override request preparer so we can get the prepared content length, for tracking transfer sizes

Return type
requests.PreparedRequest

request(*method*, *url*, ***kwargs*)

Constructs a Request, prepares it and sends it. Returns Response object.

Parameters

- **method** – method for the new Request object.
- **url** – URL for the new Request object.
- **params** – (optional) Dictionary or bytes to be sent in the query string for the Request.
- **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the Request.
- **json** – (optional) json to send in the body of the Request.
- **headers** – (optional) Dictionary of HTTP Headers to send with the Request.
- **cookies** – (optional) Dict or CookieJar object to send with the Request.
- **files** – (optional) Dictionary of `'filename': file-like-objects` for multipart encoding upload.
- **auth** – (optional) Auth tuple or callable to enable Basic/Digest/Custom HTTP Auth.
- **timeout** (*float or tuple*) – (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.
- **allow_redirects** (*bool*) – (optional) Set to True by default.
- **proxies** – (optional) Dictionary mapping protocol or protocol and hostname to the URL of the proxy.
- **stream** – (optional) whether to immediately download the response content. Defaults to False.
- **verify** – (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use. Defaults to True. When set to False, requests will accept any TLS certificate presented by the server, and will ignore hostname mismatches and/or expired certificates, which will make your application vulnerable to man-in-the-middle (MitM) attacks. Setting verify to False may be useful during local development or testing.

- **cert** – (optional) if String, path to ssl client cert file (.pem). If Tuple, ('cert', 'key') pair.

Return type
requests.Response

reset_transfer_bytes()

Resets the *bytes_sent* and *bytes_received* values to zero

The main module to be used for initiating the synchronization of data between morango instances.

class morango.sync.syncsession.**Connection**

Bases: object

Abstraction around a connection with a syncing peer (network or disk), supporting interactions with that peer. This may be used by a SyncClient, but also supports other operations (e.g. querying certificates) outside the context of syncing.

This class should be subclassed for particular transport mechanisms, and the necessary methods overridden.

class morango.sync.syncsession.**NetworkSyncConnection**(*base_url=''*, *compresslevel=9*, *retries=7*, *backoff_factor=0.3*, *chunk_size=500*)

Bases: *Connection*

base_url

property bytes_received

property bytes_sent

capabilities

certificate_signing_request(*parent_cert*, *scope_definition_id*, *scope_params*, *userargs=None*, *password=None*)

chunk_size

close()

close_sync_session(*sync_session*)

compresslevel

create_sync_session(*client_cert*, *server_cert*, *chunk_size=None*)

Starts a sync session by creating it on the server side and returning a client to use for initiating transfer operations

Parameters

- **client_cert** (Certificate) – The local certificate to use, already registered with the server
- **server_cert** (Certificate) – The server's certificate that relates to the same profile as local
- **chunk_size** (*int*) – An optional parameter specifying the size for each transferred chunk

Returns
A SyncSessionClient instance

Return type
SyncSessionClient

default_chunk_size = 500

get_remote_certificates(*primary_partition*, *scope_def_id=None*)

push_signed_client_certificate_chain(*local_parent_cert*, *scope_definition_id*, *scope_params*)

resume_sync_session(*sync_session_id*, *chunk_size=None*, *ignore_existing_process=False*)

Resumes an existing sync session given an ID

Parameters

- **sync_session_id** – The UUID of the *SyncSession* to resume
- **chunk_size** (*int*) – An optional parameter specifying the size for each transferred chunk

:param ignore_existing_process:An optional parameter specifying whether to ignore an existing active process ID

Returns
A SyncSessionClient instance

Return type
SyncSessionClient

server_info

session

urlresolve(*endpoint*, *lookup=None*)

class morango.sync.syncsession.**PullClient**(**args*, ***kwargs*)

Bases: *TransferClient*

Sync class to pull from server

context

controller

signals

sync_connection

sync_session

class morango.sync.syncsession.**PushClient**(**args*, ***kwargs*)

Bases: *TransferClient*

Sync client for pushing to a server

context

controller

signals

sync_connection

sync_session

class morango.sync.syncsession.**SyncClientSignals**(***kwargs_defaults*)

Bases: SyncSignal

Class for holding all signal types, attached to *SyncClient* as attribute. All groups are sent the *TransferSession* object via the *transfer_session* keyword argument.

dequeuing = <morango.sync.utils.SyncSignalGroup object>

Dequeuing signal group for locally or remotely dequeuing data after transfer.

queuing = <morango.sync.utils.SyncSignalGroup object>

Queuing signal group for locally or remotely queuing data before transfer.

session = <morango.sync.utils.SyncSignalGroup object>

Signal group firing for each push and pull *TransferSession*.

transferring = <morango.sync.utils.SyncSignalGroup object>

Transferring signal group for tracking progress of push/pull on *TransferSession*.

class morango.sync.syncsession.**SyncSessionClient**(*sync_connection*, *sync_session*, *controller=None*)

Bases: object

close_sync_session()

Deprecated - Please use NetworkSyncConnection.close_sync_session and NetworkSyncConnection.close

controller

get_pull_client()

returns PullClient

get_push_client()

returns PushClient

initiate_pull(*sync_filter*)

Deprecated - Please use get_pull_client and use the client :param sync_filter: Filter

initiate_push(*sync_filter*)

Deprecated - Please use get_push_client and use the client

signals

sync_connection

sync_session

class morango.sync.syncsession.**TransferClient**(*sync_connection*, *sync_session*, *controller*)

Bases: object

Base class for handling common operations for initiating syncing and other related operations.

context

controller

property current_transfer_session

finalize()

initialize(*sync_filter*)

Parameters
sync_filter – Filter

proceed_to_and_wait_for(*stage*, *error_msg=None*, *callback=None*)

Raises an exception if an ERROR result is received from calling *proceed_to_and_wait_for* :param stage: The stage to proceed to :param error_msg: An error message str to use as the exception message if it errors :param callback: A callback to pass along to the controller

run()

Execute the transferring portion of the sync

signals

sync_connection

sync_session

morango.sync.syncsession.**compress_string**(*s*, *compresslevel=9*)

7.3 Viewsets

class morango.api.viewsets.**BufferViewSet**(***kwargs*)

Bases: ListModelMixin, GenericViewSet

create(*request*)

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

pagination_class

alias of LimitOffsetPagination

parser_classes = (<class 'morango.api.parsers.GzipParser'>, <class 'rest_framework.parsers.JSONParser'>)

permission_classes = (<class 'morango.api.permissions.BufferPermissions'>,)

serializer_class

alias of BufferSerializer

class morango.api.viewsets.**CertificateChainViewSet**(***kwargs*)

Bases: ViewSet

create(*request*)

permissions = (<class 'morango.api.permissions.CertificatePushPermissions'>,)

class morango.api.viewsets.**CertificateViewSet**(***kwargs*)

Bases: CreateModelMixin, RetrieveModelMixin, ListModelMixin, GenericViewSet

authentication_classes = (<class 'morango.api.permissions.BasicMultiArgumentAuthentication'>,)

create(*request*)

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

permission_classes = (<class 'morango.api.permissions.CertificatePermissions'>,)

serializer_class

alias of CertificateSerializer

class morango.api.viewsets.**MorangoInfoViewSet**(***kwargs*)

Bases: ViewSet

retrieve(*request*, *pk=None*)

class morango.api.viewsets.**NonceViewSet**(***kwargs*)

Bases: CreateModelMixin, GenericViewSet

create(*request*)

serializer_class

alias of NonceSerializer

class morango.api.viewsets.**PublicKeyViewSet**(***kwargs*)

Bases: ReadOnlyModelViewSet

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

permission_classes = (<class 'morango.api.permissions.CertificatePushPermissions'>,)

serializer_class

alias of SharedKeySerializer

class morango.api.viewsets.**SyncSessionViewSet**(***kwargs*)

Bases: DestroyModelMixin, RetrieveModelMixin, GenericViewSet

create(*request*)

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

perform_destroy(*syncsession*)

serializer_class

alias of SyncSessionSerializer

class morango.api.viewsets.**TransferSessionViewSet**(***kwargs*)

Bases: RetrieveModelMixin, UpdateModelMixin, DestroyModelMixin, GenericViewSet

async_allowed()

Returns
A boolean if async ops are allowed by client and self

create(*request*)

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

perform_destroy(*transfer_session*)

serializer_class

alias of TransferSessionSerializer

update(*request*, **args*, ***kwargs*)

morango.api.viewsets.**controller_signal_logger**(*context=None*)

morango.api.viewsets.**get_ip**(*request*)

7.4 Permissions

class morango.api.permissions.**BasicMultiArgumentAuthentication**

Bases: BasicAuthentication

HTTP Basic authentication against username (plus any other optional arguments) and password.

authenticate_credentials(*userargs*, *password*, *request=None*)

Authenticate the userargs and password against Django auth backends. The "userargs" string may be just the username, or a querystring-encoded set of params.

class morango.api.permissions.**BufferPermissions**

Bases: BasePermission

has_permission(*request*, *view*)

Return *True* if permission is granted, *False* otherwise.

class morango.api.permissions.**CertificatePermissions**

Bases: BasePermission

has_permission(*request*, *view*)

Return *True* if permission is granted, *False* otherwise.

class morango.api.permissions.**CertificatePushPermissions**

Bases: BasePermission

has_permission(*request*, *view*)

Return *True* if permission is granted, *False* otherwise.

message = 'Server does not allow certificate pushing.'

PYTHON MODULE INDEX

m

morango.api.permissions, 52
morango.api.viewsets, 50
morango.models, 23
morango.sync.session, 46
morango.sync.syncsession, 47

INDEX

A

abstract (*morango.models.SyncableModel.Meta attribute*), 40
abstract (*morango.models.UUIDModelMixin.Meta attribute*), 45
active (*morango.models.SyncSession attribute*), 38
active (*morango.models.TransferSession attribute*), 42
as_manager() (*morango.models.SyncableModelQuerySet class method*), 41
async_allowed() (*morango.api.viewsets.TransferSessionViewSet method*), 52
authenticate_credentials() (*morango.api.permissions.BasicMultiArgumentAuthentication method*), 52
authentication_classes (*morango.api.viewsets.CertificateViewSet attribute*), 51

B

base_url (*morango.sync.syncsession.NetworkSyncConnection attribute*), 47
BasicMultiArgumentAuthentication (*class in morango.api.permissions*), 52
Buffer (*class in morango.models*), 23
Buffer.DoesNotExist, 23
Buffer.MultipleObjectsReturned, 23
buffer_set (*morango.models.TransferSession attribute*), 42
BufferPermissions (*class in morango.api.permissions*), 52
BufferViewSet (*class in morango.api.viewsets*), 50
bytes_received (*morango.models.TransferSession attribute*), 42
bytes_received (*morango.sync.session.SessionWrapper attribute*), 46
bytes_received (*morango.sync.syncsession.NetworkSyncConnection property*), 47
bytes_sent (*morango.models.TransferSession attribute*), 42
bytes_sent (*morango.sync.session.SessionWrapper attribute*), 46
bytes_sent (*morango.sync.syncsession.NetworkSyncConnection property*), 47

C

cached_clean_fields() (*morango.models.SyncableModel method*), 40
calculate_filter_specific_instance_counters() (*morango.models.DatabaseMaxCounter class method*), 28
calculate_partition() (*morango.models.SyncableModel method*), 40
calculate_source_id() (*morango.models.SyncableModel method*), 40
calculate_uuid() (*morango.models.SyncableModel method*), 41
calculate_uuid() (*morango.models.UUIDModelMixin method*), 45
capabilities (*morango.sync.syncsession.NetworkSyncConnection attribute*), 47
Certificate (*class in morango.models*), 24
Certificate.DoesNotExist, 24
Certificate.MultipleObjectsReturned, 24
certificate_set (*morango.models.Certificate attribute*), 24
certificate_set (*morango.models.ScopeDefinition attribute*), 34
certificate_signing_request() (*morango.sync.syncsession.NetworkSyncConnection method*), 47
CertificateChainViewSet (*class in morango.api.viewsets*), 50
CertificatePermissions (*class in morango.api.permissions*), 53
CertificatePushPermissions (*class in morango.api.permissions*), 53
CertificateViewSet (*class in morango.api.viewsets*), 50
check_certificate() (*morango.models.Certificate method*), 25

chunk_size (*morango.sync.syncsession.NetworkSyncConnection attribute*), 47
client_certificate (*morango.models.SyncSession attribute*), 38
client_certificate_id (*morango.models.SyncSession attribute*), 38
client_fsic (*morango.models.TransferSession attribute*), 42
client_instance_data (*morango.models.SyncSession attribute*), 38
client_instance_id (*morango.models.SyncSession attribute*), 38
client_instance_json (*morango.models.SyncSession attribute*), 38
client_ip (*morango.models.SyncSession attribute*), 38
close() (*morango.sync.syncsession.NetworkSyncConnection method*), 47
close_sync_session() (*morango.sync.syncsession.NetworkSyncConnection method*), 47
close_sync_session() (*morango.sync.syncsession.SyncSessionClient method*), 49
compress_string() (*in module morango.sync.syncsession*), 50
compresslevel (*morango.sync.syncsession.NetworkSyncConnection attribute*), 47
compute_namespaced_id() (*morango.models.SyncableModel static method*), 41
Connection (*class in morango.sync.syncsession*), 47
connection_kind (*morango.models.SyncSession attribute*), 38
connection_path (*morango.models.SyncSession attribute*), 38
contains_partition() (*morango.models.Filter method*), 29
context (*morango.sync.syncsession.PullClient attribute*), 48
context (*morango.sync.syncsession.PushClient attribute*), 48
context (*morango.sync.syncsession.TransferClient attribute*), 49
controller (*morango.sync.syncsession.PullClient attribute*), 48
controller (*morango.sync.syncsession.PushClient attribute*), 48
controller (*morango.sync.syncsession.SyncSessionClient attribute*), 49
controller (*morango.sync.syncsession.TransferClient attribute*), 49
controller_signal_logger() (*in module morango.api.viewsets*), 52
counter (*morango.models.InstanceIDModel attribute*), 30
create() (*morango.api.viewsets.BufferViewSet method*), 50
create() (*morango.api.viewsets.CertificateChainViewSet method*), 50
create() (*morango.api.viewsets.CertificateViewSet method*), 51
create() (*morango.api.viewsets.NonceViewSet method*), 51
create() (*morango.api.viewsets.SyncSessionViewSet method*), 51
create() (*morango.api.viewsets.TransferSessionViewSet method*), 52
create_sync_session() (*morango.sync.syncsession.NetworkSyncConnection method*), 47
current (*morango.models.DatabaseIDModel attribute*), 27
current (*morango.models.InstanceIDModel attribute*), 30
current (*morango.models.SharedKey attribute*), 36
current_transfer_session (*morango.sync.syncsession.TransferClient property*), 49

D

database (*morango.models.InstanceIDModel attribute*), 30
database_id (*morango.models.InstanceIDModel attribute*), 30
DatabaseIDModel (*class in morango.models*), 27
DatabaseIDModel.DoesNotExist, 27
DatabaseIDModel.MultipleObjectsReturned, 27
DatabaseMaxCounter (*class in morango.models*), 28
DatabaseMaxCounter.DoesNotExist, 28
DatabaseMaxCounter.MultipleObjectsReturned, 28
date_generated (*morango.models.DatabaseIDModel attribute*), 27
db_path (*morango.models.InstanceIDModel attribute*), 30
deconstruct() (*morango.models.UUIDField method*), 44
default_chunk_size (*morango.sync.syncsession.NetworkSyncConnection attribute*), 47
deferred_clean_fields() (*morango.models.SyncableModel method*), 41
delete() (*morango.models.SyncableModel method*), 41
delete_buffers() (*morango.models.TransferSession method*), 42
DeletedModels (*class in morango.models*), 28
DeletedModels.DoesNotExist, 29

DeletedModels.MultipleObjectsReturned, 29
dequeuing (*morango.sync.syncsession.SyncClientSignals attribute*), 49
description (*morango.models.ScopeDefinition attribute*), 34
deserialization_error (*morango.models.Store attribute*), 37
deserialize() (*morango.models.Certificate class method*), 25
deserialize() (*morango.models.SyncableModel class method*), 41
dirty_bit (*morango.models.Store attribute*), 37

E

extra_fields (*morango.models.SyncSession attribute*), 38

F

Filter (*class in morango.models*), 29
filter (*morango.models.TransferSession attribute*), 43
finalize() (*morango.sync.syncsession.TransferClient method*), 49
from_db_value() (*morango.models.PrivateKeyField method*), 32
from_db_value() (*morango.models.PublicKeyField method*), 32
from_db_value() (*morango.models.UUIDField method*), 45

G

generate_root_certificate() (*morango.models.Certificate class method*), 25
get_connection_kind_display() (*morango.models.SyncSession method*), 39
get_current_instance_and_increment_counter() (*morango.models.InstanceIDModel class method*), 30
get_db_prep_value() (*morango.models.UUIDField method*), 45
get_default() (*morango.models.UUIDField method*), 45
get_description() (*morango.models.ScopeDefinition method*), 35
get_filter() (*morango.models.TransferSession method*), 43
get_instance_counters_for_partitions() (*morango.models.DatabaseMaxCounter class method*), 28
get_internal_type() (*morango.models.UUIDField method*), 45
get_ip() (*in module morango.api.viewsets*), 52
get_next_by_date_generated() (*morango.models.DatabaseIDModel method*), 27
get_next_by_last_activity_timestamp() (*morango.models.SyncSession method*), 39
get_next_by_last_activity_timestamp() (*morango.models.TransferSession method*), 43
get_next_by_start_timestamp() (*morango.models.SyncSession method*), 39
get_next_by_start_timestamp() (*morango.models.TransferSession method*), 43
get_next_by_timestamp() (*morango.models.Nonce method*), 31
get_or_create_current_database_id() (*morango.models.DatabaseIDModel class method*), 27
get_or_create_current_instance() (*morango.models.InstanceIDModel class method*), 30
get_or_create_shared_key() (*morango.models.SharedKey class method*), 36
get_prep_value() (*morango.models.PrivateKeyField method*), 32
get_prep_value() (*morango.models.PublicKeyField method*), 32
get_previous_by_date_generated() (*morango.models.DatabaseIDModel method*), 27
get_previous_by_last_activity_timestamp() (*morango.models.SyncSession method*), 39
get_previous_by_last_activity_timestamp() (*morango.models.TransferSession method*), 43
get_previous_by_start_timestamp() (*morango.models.SyncSession method*), 39
get_previous_by_start_timestamp() (*morango.models.TransferSession method*), 43
get_previous_by_timestamp() (*morango.models.Nonce method*), 32
get_proquint() (*morango.models.InstanceIDModel method*), 31
get_pull_client() (*morango.sync.syncsession.SyncSessionClient method*), 49
get_push_client() (*morango.sync.syncsession.SyncSessionClient method*), 49
get_queryset() (*morango.api.viewsets.BufferViewSet method*), 50
get_queryset() (*morango.api.viewsets.CertificateViewSet method*), 51
get_queryset() (*morango.api.viewsets.PublicKeyViewSet method*), 51
get_queryset() (*morango.api.viewsets.SyncSessionViewSet method*), 51
get_queryset() (*morango.api.viewsets.TransferSessionViewSet method*), 52

get_remote_certificates() (*morango.sync.syncsession.NetworkSyncConnection method*), 48
get_scope() (*morango.models.Certificate method*), 25
get_scope() (*morango.models.ScopeDefinition method*), 35
get_touched_record_ids_for_model() (*morango.models.TransferSession method*), 43
get_transfer_stage_display() (*morango.models.TransferSession method*), 43
get_transfer_stage_status_display() (*morango.models.TransferSession method*), 43

H

HardDeletedModels (*class in morango.models*), 29
HardDeletedModels.DoesNotExist, 29
HardDeletedModels.MultipleObjectsReturned, 29
has_permission() (*morango.api.permissions.BufferPermissions method*), 53
has_permission() (*morango.api.permissions.CertificatePermissions method*), 53
has_permission() (*morango.api.permissions.CertificatePushPermissions method*), 53
has_private_key() (*morango.models.Certificate method*), 25
hostname (*morango.models.InstanceIDModel attribute*), 31

I

id (*morango.models.Buffer attribute*), 23
id (*morango.models.DatabaseMaxCounter attribute*), 28
id (*morango.models.DeletedModels attribute*), 29
id (*morango.models.HardDeletedModels attribute*), 29
id (*morango.models.InstanceIDModel attribute*), 31
id (*morango.models.RecordMaxCounter attribute*), 33
id (*morango.models.RecordMaxCounterBuffer attribute*), 33
id (*morango.models.ScopeDefinition attribute*), 35
id (*morango.models.SharedKey attribute*), 36
id (*morango.models.Store attribute*), 37
id (*morango.models.SyncSession attribute*), 39
id (*morango.models.TransferSession attribute*), 43
id (*morango.models.UUIDModelMixin attribute*), 45
ID_PLACEHOLDER (*morango.models.SyncableModel attribute*), 40
initial_instance_id (*morango.models.DatabaseIDModel attribute*), 27
initialize() (*morango.sync.syncsession.TransferClient method*), 49
initiate_pull() (*morango.sync.syncsession.SyncSessionClient method*), 49
initiate_push() (*morango.sync.syncsession.SyncSessionClient method*), 49
instance_info (*morango.models.InstanceIDModel property*), 31
InstanceIDModel (*class in morango.models*), 29
InstanceIDModel.DoesNotExist, 30
InstanceIDModel.MultipleObjectsReturned, 30
instanceidmodel_set (*morango.models.DatabaseIDModel attribute*), 27
ip (*morango.models.Nonce attribute*), 32
is_server (*morango.models.SyncSession attribute*), 39
is_subset_of() (*morango.models.Filter method*), 29
is_subset_of() (*morango.models.Scope method*), 34

L

last_activity_timestamp (*morango.models.SyncSession attribute*), 39
last_activity_timestamp (*morango.models.TransferSession attribute*), 43
last_transfer_session_id (*morango.models.Store attribute*), 37
level (*morango.models.Certificate attribute*), 25
lft (*morango.models.Certificate attribute*), 25

M

merge_conflict() (*morango.models.SyncableModel class method*), 41
message (*morango.api.permissions.CertificatePushPermissions attribute*), 53
model_uuid (*morango.models.Buffer attribute*), 23
model_uuid (*morango.models.RecordMaxCounterBuffer attribute*), 33
module
    morango.api.permissions, 52
    morango.api.viewsets, 50
    morango.models, 23
    morango.sync.session, 46
    morango.sync.syncsession, 47
morango.api.permissions
    module, 52
morango.api.viewsets
    module, 50
morango.models
    module, 23
morango.sync.session
    module, 46
morango.sync.syncsession
    module, 47

morango_fields_not_to_serialize (*morango.models.SyncableModel attribute*), 41
morango_model_dependencies (*morango.models.SyncableModel attribute*), 41
morango_profile (*morango.models.SyncableModel attribute*), 41
MorangoInfoViewSet (*class in morango.api.viewsets*), 51

N

NetworkSyncConnection (*class in morango.sync.syncsession*), 47
node_id (*morango.models.InstanceIDModel attribute*), 31
Nonce (*class in morango.models*), 31
Nonce.DoesNotExist, 31
Nonce.MultipleObjectsReturned, 31
NonceViewSet (*class in morango.api.viewsets*), 51

O

objects (*morango.models.Buffer attribute*), 24
objects (*morango.models.DatabaseIDModel attribute*), 28
objects (*morango.models.DatabaseMaxCounter attribute*), 28
objects (*morango.models.DeletedModels attribute*), 29
objects (*morango.models.HardDeletedModels attribute*), 29
objects (*morango.models.InstanceIDModel attribute*), 31
objects (*morango.models.Nonce attribute*), 32
objects (*morango.models.RecordMaxCounter attribute*), 33
objects (*morango.models.RecordMaxCounterBuffer attribute*), 33
objects (*morango.models.ScopeDefinition attribute*), 35
objects (*morango.models.SharedKey attribute*), 36
objects (*morango.models.Store attribute*), 37
objects (*morango.models.SyncableModel attribute*), 41
objects (*morango.models.SyncSession attribute*), 39
objects (*morango.models.TransferSession attribute*), 43

P

pagination_class (*morango.api.viewsets.BufferViewSet attribute*), 50
parent (*morango.models.Certificate attribute*), 25
parent_id (*morango.models.Certificate attribute*), 25
parser_classes (*morango.api.viewsets.BufferViewSet attribute*), 50
partition (*morango.models.DatabaseMaxCounter attribute*), 28
perform_destroy() (*morango.api.viewsets.SyncSessionViewSet method*), 52
perform_destroy() (*morango.api.viewsets.TransferSessionViewSet method*), 52
permission_classes (*morango.api.viewsets.BufferViewSet attribute*), 50
permission_classes (*morango.api.viewsets.CertificateViewSet attribute*), 51
permission_classes (*morango.api.viewsets.PublicKeyViewSet attribute*), 51
permissions (*morango.api.viewsets.CertificateChainViewSet attribute*), 50
platform (*morango.models.InstanceIDModel attribute*), 31
prepare_request() (*morango.sync.session.SessionWrapper method*), 46
prepare_value() (*morango.models.UUIDField method*), 45
primary_scope_param_key (*morango.models.ScopeDefinition attribute*), 35
private_key (*morango.models.Certificate property*), 25
private_key (*morango.models.SharedKey attribute*), 36
PrivateKeyField (*class in morango.models*), 32
proceed_to_and_wait_for() (*morango.sync.syncsession.TransferClient method*), 50
process_id (*morango.models.SyncSession attribute*), 39
profile (*morango.models.Certificate attribute*), 25
profile (*morango.models.DeletedModels attribute*), 29
profile (*morango.models.HardDeletedModels attribute*), 29
profile (*morango.models.ScopeDefinition attribute*), 35
profile (*morango.models.SyncSession attribute*), 39
public_key (*morango.models.Certificate attribute*), 25
public_key (*morango.models.SharedKey attribute*), 36
PublicKeyField (*class in morango.models*), 32
PublicKeyViewSet (*class in morango.api.viewsets*), 51
pull (*morango.models.TransferSession property*), 43
PullClient (*class in morango.sync.syncsession*), 48
push (*morango.models.TransferSession attribute*), 43
push_signed_client_certificate_chain() (*morango.sync.syncsession.NetworkSyncConnection method*), 48
PushClient (*class in morango.sync.syncsession*), 48

Q

queuing (*morango.sync.syncsession.SyncClientSignals attribute*), 49

R

read_filter_template (*morango.models.ScopeDefinition attribute*), 35
read_write_filter_template (*morango.models.ScopeDefinition attribute*),

35
RecordMaxCounter (*class in morango.models*), 32
RecordMaxCounter.DoesNotExist, 32
RecordMaxCounter.MultipleObjectsReturned, 33
recordmaxcounter_set (*morango.models.Store attribute*), 37
RecordMaxCounterBuffer (*class in morango.models*), 33
RecordMaxCounterBuffer.DoesNotExist, 33
RecordMaxCounterBuffer.MultipleObjectsReturned, 33
recordmaxcounterbuffer_set (*morango.models.TransferSession attribute*), 43
records_total (*morango.models.TransferSession attribute*), 43
records_transferred (*morango.models.TransferSession attribute*), 44
request() (*morango.sync.session.SessionWrapper method*), 46
reset_transfer_bytes() (*morango.sync.session.SessionWrapper method*), 47
resume_sync_session() (*morango.sync.syncsession.NetworkSyncConnection method*), 48
retrieve() (*morango.api.viewsets.MorangoInfoViewSet method*), 51
retrieve_by_id() (*morango.models.ScopeDefinition class method*), 35
rght (*morango.models.Certificate attribute*), 25
rmcb_list() (*morango.models.Buffer method*), 24
run() (*morango.sync.syncsession.TransferClient method*), 50

S

salt (*morango.models.Certificate attribute*), 25
save() (*morango.models.DatabaseIDModel method*), 28
save() (*morango.models.SyncableModel method*), 41
save() (*morango.models.UUIDModelMixin method*), 45
save_certificate_chain() (*morango.models.Certificate class method*), 26
Scope (*class in morango.models*), 34
scope_definition (*morango.models.Certificate attribute*), 26
scope_definition_id (*morango.models.Certificate attribute*), 26
scope_params (*morango.models.Certificate attribute*), 26
scope_version (*morango.models.Certificate attribute*), 26
ScopeDefinition (*class in morango.models*), 34
ScopeDefinition.DoesNotExist, 34
ScopeDefinition.MultipleObjectsReturned, 34
serialize() (*morango.models.Certificate method*), 26
serialize() (*morango.models.SyncableModel method*), 41
serialized (*morango.models.Certificate attribute*), 26
serializer_class (*morango.api.viewsets.BufferViewSet attribute*), 50
serializer_class (*morango.api.viewsets.CertificateViewSet attribute*), 51
serializer_class (*morango.api.viewsets.NonceViewSet attribute*), 51
serializer_class (*morango.api.viewsets.PublicKeyViewSet attribute*), 51
serializer_class (*morango.api.viewsets.SyncSessionViewSet attribute*), 52
serializer_class (*morango.api.viewsets.TransferSessionViewSet attribute*), 52
server_certificate (*morango.models.SyncSession attribute*), 39
server_certificate_id (*morango.models.SyncSession attribute*), 39
server_fsic (*morango.models.TransferSession attribute*), 44
server_info (*morango.sync.syncsession.NetworkSyncConnection attribute*), 48
server_instance_data (*morango.models.SyncSession attribute*), 39
server_instance_id (*morango.models.SyncSession attribute*), 39
server_instance_json (*morango.models.SyncSession attribute*), 39
server_ip (*morango.models.SyncSession attribute*), 40
session (*morango.sync.syncsession.NetworkSyncConnection attribute*), 48
session (*morango.sync.syncsession.SyncClientSignals attribute*), 49
SessionWrapper (*class in morango.sync.session*), 46
SharedKey (*class in morango.models*), 35
SharedKey.DoesNotExist, 35
SharedKey.MultipleObjectsReturned, 35
sign() (*morango.models.Certificate method*), 26
sign_certificate() (*morango.models.Certificate method*), 26
signals (*morango.sync.syncsession.PullClient attribute*), 48
signals (*morango.sync.syncsession.PushClient attribute*), 48
signals (*morango.sync.syncsession.SyncSessionClient attribute*), 49
signals (*morango.sync.syncsession.TransferClient attribute*), 50
signature (*morango.models.Certificate attribute*), 26
start_timestamp (*morango.models.SyncSession*

attribute), 40
start_timestamp (morango.models.TransferSession attribute), 44
Store (class in morango.models), 36
Store.DoesNotExist, 36
Store.MultipleObjectsReturned, 36
store_model (morango.models.RecordMaxCounter attribute), 33
store_model_id (morango.models.RecordMaxCounter attribute), 33
sync_connection (morango.sync.syncsession.PullClient attribute), 48
sync_connection (morango.sync.syncsession.PushClient attribute), 48
sync_connection (morango.sync.syncsession.SyncSessionClient attribute), 49
sync_connection (morango.sync.syncsession.TransferClient attribute), 50
sync_session (morango.models.TransferSession attribute), 44
sync_session (morango.sync.syncsession.PullClient attribute), 48
sync_session (morango.sync.syncsession.PushClient attribute), 48
sync_session (morango.sync.syncsession.SyncSessionClient attribute), 49
sync_session (morango.sync.syncsession.TransferClient attribute), 50
sync_session_id (morango.models.TransferSession attribute), 44
SyncableModel (class in morango.models), 40
SyncableModel.Meta (class in morango.models), 40
SyncableModelManager (class in morango.models), 41
SyncableModelQuerySet (class in morango.models), 41
SyncClientSignals (class in morango.sync.syncsession), 48
SyncSession (class in morango.models), 37
SyncSession.DoesNotExist, 38
SyncSession.MultipleObjectsReturned, 38
SyncSessionClient (class in morango.sync.syncsession), 49
syncsessions_client (morango.models.Certificate attribute), 26
syncsessions_server (morango.models.Certificate attribute), 26
SyncSessionViewSet (class in morango.api.viewsets), 51
system_id (morango.models.InstanceIDModel attribute), 31
sysversion (morango.models.InstanceIDModel attribute), 31

T

timestamp (morango.models.Nonce attribute), 32
to_python() (morango.models.PrivateKeyField method), 32
to_python() (morango.models.PublicKeyField method), 32
to_python() (morango.models.UUIDField method), 45
transfer_session (morango.models.Buffer attribute), 24
transfer_session (morango.models.RecordMaxCounterBuffer attribute), 33
transfer_session_id (morango.models.Buffer attribute), 24
transfer_session_id (morango.models.RecordMaxCounterBuffer attribute), 34
transfer_stage (morango.models.TransferSession attribute), 44
transfer_stage_status (morango.models.TransferSession attribute), 44
TransferClient (class in morango.sync.syncsession), 49
transferring (morango.sync.syncsession.SyncClientSignals attribute), 49
TransferSession (class in morango.models), 41
TransferSession.DoesNotExist, 42
TransferSession.MultipleObjectsReturned, 42
transfersession_set (morango.models.SyncSession attribute), 40
TransferSessionViewSet (class in morango.api.viewsets), 52
tree_id (morango.models.Certificate attribute), 27

U

update() (morango.api.viewsets.TransferSessionViewSet method), 52
update() (morango.models.SyncableModelQuerySet method), 41
update_fsics() (morango.models.DatabaseMaxCounter class method), 28
update_state() (morango.models.TransferSession method), 44
urlresolve() (morango.sync.syncsession.NetworkSyncConnection method), 48
use_nonce() (morango.models.Nonce class method), 32
uuid_input_fields (morango.models.Certificate attribute), 27
uuid_input_fields (morango.models.DatabaseIDModel attribute), 28
uuid_input_fields (morango.models.InstanceIDModel attribute), 31
uuid_input_fields (morango.models.Nonce attribute), 32

uuid_input_fields (*morango.models.UUIDModelMixin attribute*), 45
UUIDField (*class in morango.models*), 44
UUIDModelMixin (*class in morango.models*), 45
UUIDModelMixin.Meta (*class in morango.models*), 45

V

verify() (*morango.models.Certificate method*), 27
version (*morango.models.ScopeDefinition attribute*), 35

W

write_filter_template (*morango.models.ScopeDefinition attribute*), 35