Morango is a Django database replication engine written in pure Python. It is designed and maintained in support of the Kolibri product ecosystem.
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.
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",
    (continues on next page)
```
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.
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.
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
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.
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 \( A_1 \) and history \([ A_1 ]\).
2. Next, *Device A* modifies \( r \). The record version changes to \( A_2 \) and the history is now \([ A_2, A_1 ]\).
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 \( B_1 \). The history of \( r \) is now \([ B_1, A_2, A_1 ]\) on *Device B* and still \([ A_2, A_1 ]\) on *Device A*.
5. When *Device A* syncs with *Device B* again (the arrow), there is no conflict and the update \( B_1 \) 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 \( A_1 \) and history \([ A_1 ]\).
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 \( B_1 \) and the history \([ B_1, A_1 ]\).
4. *Device A* modifies its own copy of record \( r \) and saves it as \( A_2 \) with history \([ A_2, A_1 ]\).
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.
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.
7.1 Models

class morango.models.UUIDField(*args, **kwargs)
    Bases: django.db.models.fields.CharField

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

deconstruct ()
    Returns 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: 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, unicode, int, long, 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.

```python
from_db_value(value, expression, connection, context)
```

```python
get_db_prep_value(value, connection, prepared=False)
```

Returns field’s value prepared for interacting with the database backend.

Used by the default implementations of `get_db_prep_save()`.

```python
get_default()
```

Returns the default value for this field.

```python
get_internal_type()
```

```python
prepare_value(value)
```

```python
to_python(value)
```

Converts the input value into the expected Python data type, raising `django.core.exceptions.ValidationError` if the data can’t be converted. Returns the converted value. Subclasses should override this.

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

Bases: `morango.models.fields.crypto.RSAKeyBaseField`

```python
from_db_value(value, expression, connection, context)
```

```python
get_prep_value(value)
```

Perform preliminary non-db specific value checks and conversions.

```python
to_python(value)
```

Converts the input value into the expected Python data type, raising `django.core.exceptions.ValidationError` if the data can’t be converted. Returns the converted value. Subclasses should override this.

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

Bases: `morango.models.fields.crypto.RSAKeyBaseField`

```python
from_db_value(value, expression, connection, context)
```

```python
get_prep_value(value)
```

Perform preliminary non-db specific value checks and conversions.

```python
to_python(value)
```

Converts the input value into the expected Python data type, raising `django.core.exceptions.ValidationError` if the data can’t be converted. Returns the converted value. Subclasses should override this.

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

Bases: `django.db.models.base.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

```python
exception DoesNotExist
```

Bases: `django.core.exceptions.ObjectDoesNotExist`
exception MultipleObjectsReturned
    Bases: django.core.exceptions.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.UUIDModelMixin(*args, **kwargs)
    Bases: django.db.models.base.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)
    Saves 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

class morango.models.Certificate(id, parent, profile, scope_definition, scope_version, scope_params, public_key, salt, serialized, signature, _private_key)
    Bases: mptt.models.MPTTModel,morango.models.fields.uuids.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

eception DoesNotExist
    Bases: django.core.exceptions.ObjectDoesNotExist

eception MultipleObjectsReturned
    Bases: django.core.exceptions.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 ReverseManyToManyDescriptor 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:
The code snippet is:

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

child.parent is a ForwardManyToOneDescriptor instance.

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

**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.

**right**
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 ForwardToOneToOneDescriptor subclass) relation.

In the example:

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

child.parent is a ForwardManyToOneDescriptor instance.

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

**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:

```python
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:

```python
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)

```python
class morango.models.Nonce(*args, **kwargs)
    Bases: morango.models.fields.uuids.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: django.core.exceptions.ObjectDoesNotExist

**exception** MultipleObjectsReturned
    Bases: django.core.exceptions.MultipleObjectsReturned

**get_next_by_timestamp**(**morekwargs**)

**get_previous_by_timestamp**(**morekwargs**)
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.ScopeDefinition(id, profile, version, primary_scope_param_key, description, read_filter_template, write_filter_template, read_write_filter_template)

Bases: django.db.models.base.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: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.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 ReverseManyToManyDescriptor 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.Filter(template, params={})
  Bases: object
  contains_partition(partition)
  is_subset_of(other)

**class** morango.models.Scope(definition, params)
  Bases: object
  is_subset_of(other)

**class** morango.models.SyncableModelManager
  Bases: django.db.models.manager.ManagerFromSyncableModelQuerySet

**class** morango.models.SyncableModelQuerySet(model=None, query=None, using=None, hints=None)
  Bases: django.db.models.query.QuerySet
  classmethod as_manager()
  update(update_dirty_bit_to=True, **kwargs)
  Updates all elements in the current QuerySet, setting all the given fields to the appropriate values.

**class** morango.models.MorangoTreeQuerySet(model=None, query=None, using=None, hints=None)
  Bases: mptt.querysets.TreeQuerySet, morango.models.query.SyncableModelQuerySet

**class** morango.models.MorangoMPTTTreeManager
  Bases: mptt.managers.TreeManager,morango.models.manager.SyncableModelManager
get_queryset()
Ensures that this manager always returns nodes in tree order.

class morango.models.MorangoMPTTModel(*args, **kwargs)
Bases: mptt.models.MPTTModel
Any model that inherits from SyncableModel that also wants to inherit from MPTTModel should instead inherit from MorangoMPTTModel, which modifies some behavior to make it safe for the syncing system.

class Meta
    Bases: object
    abstract = False

objects
class morango.models.DatabaseIDModel(*args, **kwargs)
Bases: morango.models.fields.uuids.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: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
    Bases: django.core.exceptions.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(**morekwargs)
classmethod get_or_create_current_database_id()
get_previous_by_date_generated(**morekwargs)

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 ReverseManyToManyDescriptor 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>`

`saves (*args, **kwargs)`
Saves 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.InstanceIDModel (*args, **kwargs)`
Bases: `django.db.models.base.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: `django.core.exceptions.ObjectDoesNotExist`

`exception MultipleObjectsReturned`
Bases: `django.core.exceptions.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:

```python
class Child (Model):
    parent = ForeignKey(Parent, related_name='children')
```
child.parent is a ForwardManyToOneDescriptor instance.

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

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.

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.SyncSession(*args, **kwargs)
Bases: django.db.models.base.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

7.1. Models
- 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 (TextField) – Client instance
- server_instance (TextField) – Server instance
- extra_fields (TextField) – Extra fields

exception DoesNotExist
    Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
    Bases: django.core.exceptions.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 ForwardOneToManyDescriptor subclass) relation.
    In the example:

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

child.parent is a ForwardManyToOneDescriptor instance.

client_certificate_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
    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(**morekwargs)
get_next_by_last_activity_timestamp(**morekwargs)
get_next_by_start_timestamp(**morekwargs)
get_previous_by_last_activity_timestamp(**morekwargs)
get_previous_by_start_timestamp(**morekwargs)

**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>

**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:

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

child.parent is a ForwardManyToManyDescriptor instance.

**server_certificate_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**
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:

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

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

```python
class morango.models.TransferSession(*args, **kwargs)
    Bases: django.db.models.base.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

exception DoesNotExist
    Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
    Bases: django.core.exceptions.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:
```

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

parent.children is a ReverseManyToManyDescriptor 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.

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(**morekwargs)
get_next_by_start_timestamp(**morekwargs)
get_previous_by_last_activity_timestamp(**morekwargs)
get_previous_by_start_timestamp(**morekwargs)

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>
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 ReverseManyToManyDescriptor 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:

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

child.parent is a ForwardManyToOneDescriptor instance.

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

```python
class morango.models.DeletedModels(*args, **kwargs)
Bases: django.db.models.base.Model

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

**Parameters**

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

**exception DoesNotExist**
Bases: django.core.exceptions.ObjectDoesNotExist

**exception MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

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

```python
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.

```python
class morango.models.HardDeletedModels(*args, **kwargs)
Bases: django.db.models.base.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: django.core.exceptions.ObjectDoesNotExist

**exception MultipleObjectsReturned**
Bases: django.core.exceptions.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.Store(*args, **kwargs)
    Bases: morango.models.core.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

exception DoesNotExist
    Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
    Bases: django.core.exceptions.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.

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

7.1. Models
**recordmaxcounter_set**

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

In the example:

```python
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.

```python
class morango.models.Buffer(*args, **kwargs)
Bases: morango.models.core.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: django.core.exceptions.ObjectDoesNotExist

**exception MultipleObjectsReturned**

Bases: django.core.exceptions.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:

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

child.parent is a ForwardManyToManyDescriptor instance.

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.

```python
class morango.models.DatabaseMaxCounter(*args, **kwargs):
    Bases: morango.models.core.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
```

```python
exception DoesNotExist
    Bases: django.core.exceptions.ObjectDoesNotExist
```

```python
exception MultipleObjectsReturned
    Bases: django.core.exceptions.MultipleObjectsReturned
```

```python
classmethod calculate_filter_max_counters(filters)
```

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

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

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

```python
classmethod update_fsics(fsics, sync_filter)
```

```python
class morango.models.RecordMaxCounter(*args, **kwargs):
    Bases: morango.models.core.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: `django.core.exceptions.ObjectDoesNotExist`

`exception MultipleObjectsReturned`

Bases: `django.core.exceptions.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:

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

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`store_model_id`

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.RecordMaxCounterBuffer(*args, **kwargs)`

Bases: `morango.models.core.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: `django.core.exceptions.ObjectDoesNotExist`

`exception MultipleObjectsReturned`

Bases: `django.core.exceptions.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:

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

child.parent is a ForwardManyToManyDescriptor instance.

**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.

```python
class morango.models.SyncableModel(*args, **kwargs)
Bases: morango.models.fields.uuids.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)
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)
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*)

Saves 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.

### 7.2 Sync sessions

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

*Bases:* requests.sessions.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

**Parameters**

- **request** – requests.Request

**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

```python
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.

```python
class morango.sync.syncsession.BaseSyncClient (sync_connection=sync_connection, sync_session=sync_session, chunk_size=500):
    Bases: object

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

    finalize (allow_server_timeout=False)

    initialize (sync_filter)

    Parameters
    
    sync_filter – Filter

    run ()

    signals = None

    A namespaced attribute clearly separating signal groups

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):
    Bases: morango.sync.syncsession.Connection

    bytes_received

    bytes_sent

    certificateSigning_request (parent_cert, scope_definition_id, scope_params, user-args=None, password=None)

    close ()

    close_sync_session (sync_session)

    create_sync_session (client_cert, server_cert, 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)
```

**7.2. Sync sessions**
urlresolve (endpoint, lookup=None)

class morango.sync.syncsession.PullClient (sync_connection,  
sync_session,  
chunk_size=500)

Bases: morango.sync.syncsession.BaseSyncClient

Sync class to pull from server

finalize (allow_server_timeout=False)

initialize (sync_filter)

Parameters sync_filter – Filter

r

sync_filter = None

class morango.sync.syncsession.PushClient (sync_connection,  
sync_session,  
chunk_size=500)

Bases: morango.sync.syncsession.BaseSyncClient

Sync client for pushing to a server

finalize (allow_server_timeout=False)

initialize (sync_filter)

Parameters sync_filter – Filter

r

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

Bases: morango.sync.utils.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.

dequeueing = <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,  
chunk_size=500)

Bases: object

close_sync_session ()

Deprecated - Please use NetworkSyncConnection.close_sync_session and NetworkSyncConnection.close

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 = None
   A namespaced attribute clearly separating signal groups

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

7.3 Viewsets

7.4 Permissions

class morango.api.permissions.BasicMultiArgumentAuthentication
   Bases: rest_framework.authentication.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: rest_framework.permissions.BasePermission

   has_permission(request, view)
      Return True if permission is granted, False otherwise.

class morango.api.permissions.CertificatePermissions
   Bases: rest_framework.permissions.BasePermission

   has_permission(request, view)
      Return True if permission is granted, False otherwise.

class morango.api.permissions.CertificatePushPermissions
   Bases: rest_framework.permissions.BasePermission

   has_permission(request, view)
      Return True if permission is granted, False otherwise.

   message = 'Server does not allow certificate pushing.'

class morango.api.permissions.NoncePermissions
   Bases: rest_framework.permissions.BasePermission

   has_permission(request, view)
      Return True if permission is granted, False otherwise.

class morango.api.permissions.SyncSessionPermissions
   Bases: rest_framework.permissions.BasePermission

   has_permission(request, view)
      Return True if permission is granted, False otherwise.

class morango.api.permissions.TransferSessionPermissions
   Bases: rest_framework.permissions.BasePermission

   has_permission(request, view)
      Return True if permission is granted, False otherwise.
Python Module Index

m

morango.api.permissions, 45
morango.models, 21
morango.sync.session, 42
morango.sync.syncsession, 43
# Index

A
- abstract (morango.models.MorangoMPTTModel.Meta attribute), 29
- abstract (morango.models.SyncableModel.Meta attribute), 41
- abstract (morango.models.UUIDModelMixin.Meta attribute), 23
- active (morango.models.SyncSession attribute), 32
- active (morango.models.TransferSession attribute), 34
- as_manager() (morango.models.SyncableModelQuerySet class method), 28
- authenticate_credentials() (morango.api.permissions.BasicMultiArgumentAuthentication method), 45

B
- BaseSyncClient (class in morango.sync.syncsession), 43
- BasicMultiArgumentAuthentication (class in morango.api.permissions), 45
- Buffer (class in morango.models), 38
- Buffer.DoesNotExist, 38
- Buffer.MultipleObjectsReturned, 38
- buffer_set (morango.models.TransferSession attribute), 34
- BufferPermissions (class in morango.api.permissions), 45
- bytes_received (morango.models.TransferSession attribute), 34
- bytes_received (morango.sync.session.SessionWrapper attribute), 42
- bytes_received (morango.sync.syncsession.NetworkSyncConnection attribute), 43
- bytes_sent (morango.models.TransferSession attribute), 34
- bytes_sent (morango.sync.session.SessionWrapper attribute), 42
- bytes_sent (morango.sync.syncsession.NetworkSyncConnection attribute), 43

C
- cached_clean_fields() (morango.models.SyncableModel method), 41
- calculate_filter_max_counters() (morango.models.DatabaseMaxCounter class method), 39
- calculate_partition() (morango.models.SyncableModel method), 41
- calculate_source_id() (morango.models.SyncableModel method), 41
- calculate_uuid() (morango.models.SyncableModel method), 41
- authenticate() (morango.models.UUIDModelMixin method), 23
- Certificate (class in morango.models), 23
- Certificate.DoesNotExist, 24
- Certificate.MultipleObjectsReturned, 24
- certificate_set (morango.models.Certificate attribute), 24
- certificate_set (morango.models.ScopeDefinition attribute), 27
- certificate_signing_request() (morango.sync.syncsession.NetworkSyncConnection method), 43
- CertificatePermissions (class in morango.api.permissions), 45
- CertificatePushPermissions (class in morango.api.permissions), 45
- check_certificate() (morango.models.Certificate method), 24
- check_certificate() (morango.models.SyncSession attribute), 32
- client_certificate_id (morango.models.SyncSession attribute), 32
- client_fsic (morango.models.TransferSession attribute), 32
- client_instance (morango.models.SyncSession attribute), 35
- client_ip (morango.models.SyncSession attribute), 32
- close() (morango.sync.syncsession.NetworkSyncConnection method), 49
E
SyncConnection
extra_fields (morango.models.SyncSession attribute), 32
Filter (class in morango.models), 28
finalize() (morango.sync.syncsession.BaseSyncClient method), 43
finalize() (morango.sync.syncsession.PullClient method), 44
finalize() (morango.sync.syncsession.PushClient method), 44
der_from_db_value() (morango.models.PrivateKeyField method), 22
d_from_db_value() (morango.models.PublicKeyField method), 22
d_from_db_value() (morango.models.UUIDField method), 22
generate_root_certificate() (morango.models.Certificate class method), 24
get_connection_kind_display() (morango.models.SyncSession method), 32
generate_root_certificate() (morango.models.Certificate class method), 24
generate_root_certificate() (morango.models.Certificate class method), 24
generate_root_certificate() (morango.models.Certificate class method), 24
generate_root_certificate() (morango.models.Certificate class method), 24
get_description() (morango.models.ScopeDefinition method), 27
generate_root_certificate() (morango.models.Certificate class method), 24
generate_root_certificate() (morango.models.Certificate class method), 24
get_db_prep_value() (morango.models.UUIDField method), 31
generate_root_certificate() (morango.models.Certificate class method), 24
generate_root_certificate() (morango.models.Certificate class method), 24
get_default() (morango.models.UUIDField method), 22
generate_root_certificate() (morango.models.Certificate class method), 24
generate_root_certificate() (morango.models.Certificate class method), 24
get_default() (morango.models.UUIDField method), 22
generate_root_certificate() (morango.models.Certificate class method), 24
generate_root_certificate() (morango.models.Certificate class method), 24
get_default() (morango.models.UUIDField method), 22
generate_root_certificate() (morango.models.Certificate class method), 24
generate_root_certificate() (morango.models.Certificate class method), 24
get_default() (morango.models.UUIDField method), 22
generate_root_certificate() (morango.models.Certificate class method), 24
generate_root_certificate() (morango.models.Certificate class method), 24
get_default() (morango.models.UUIDField method), 22

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
dirty_bit (morango.models.Store attribute), 37

dirty_bit (morango.models.Store attribute), 37
method), 29

get_or_create_current_instance() (morango.models.InstanceIDModel class method), 31

get_or_create_shared_key() (morango.models.SharedKey class method), 23

get_prep_value() (morango.models.PrivateKeyField method), 22

get_prep_value() (morango.models.PublicKeyField method), 22

get_previous_by_date_generated() (morango.models.DatabaseIDModel method), 29

get_previous_by_last_activity_timestamp() (morango.models.SyncSession method), 33

get_previous_by_last_activity_timestamp() (morango.models.TransferSession method), 35

get_previous_by_start_timestamp() (morango.models.SyncSession method), 33

get_previous_by_start_timestamp() (morango.models.TransferSession method), 35

get_previous_by_timestamp() (morango.models.Nonce method), 26

get_proquint() (morango.models.InstanceIDModel method), 31

has_permission() (morango.api.permissions.TransferSessionPermissions method), 45

hostname (morango.models.InstanceIDModel attribute), 31

id (morango.models.Buffer attribute), 38

id (morango.models.DatabaseMaxCounter attribute), 39

id (morango.models.DeletedModels attribute), 36

id (morango.models.HardDeletedModels attribute), 36

id (morango.models.Store attribute), 37

ID_PLACEHOLDER (morango.models.SyncableModel attribute), 41

initial_instance_id (morango.models.DatabaseIDModel attribute), 29

initialize() (morango.sync.syncsession.BaseSyncClient method), 43

initialize() (morango.sync.syncsession.TransferSyncClient method), 36

initialize() (morango.sync.syncsession.PullSyncClient method), 44

initialize() (morango.sync.syncsession.PullSyncClient method), 44

initialize() (morango.sync.syncsession.PullSyncClient method), 44

initialize() (morango.sync.syncsession.PushSyncClient method), 44

initialize() (morango.sync.syncsession.PushSyncClient method), 44

initialize() (morango.sync.syncsession.PullSyncClient method), 44

initialize() (morango.sync.syncsession.PullSyncClient method), 44

initialize() (morango.sync.syncsession.PullSyncClient method), 44

initialize() (morango.sync.syncsession.PullSyncClient method), 44

initial_instance_id (morango.models.DatabaseIDModel attribute), 29

is_subset_of() (morango.models.Filter attribute), 28

is_subset_of() (morango.models.Certificate attribute), 24

Index 51
merge_conflict() (morango.models.SyncableModel class method), 41
message (morango.api.permissions.CertificatePushPermissions attribute), 45
model_uuid (morango.models.Buffer attribute), 38
model_uuid (morango.models.RecordMaxCounterBuffer attribute), 40
morango.api.permissions (module), 45
morango.models (module), 21
morango.sync.session (module), 42
morango.sync.syncsession (module), 43
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
MorangoMPTTTModel (class in morango.models), 29
MorangoMPTTTModel.Meta (class in morango.models), 29
MorangoMPTTTreeManager (class in morango.models), 28
MorangoTreeQuerySet (class in morango.models), 28

NetworkSyncConnection (class in morango.sync.syncsession), 43
node_id (morango.models.InstanceIDModel attribute), 31
Nonce (class in morango.models), 26
Nonce.DoesNotExist, 26
Nonce.MultipleObjectsReturned, 26
NoncePermissions (class in morango.api.permissions), 45

objects (morango.models.Buffer attribute), 38
objects (morango.models.DatabaseIDModel attribute), 30
objects (morango.models.DatabaseMaxCounter attribute), 39
objects (morango.models.DeletedModels attribute), 36
objects (morango.models.HardDeletedModels attribute), 37
objects (morango.models.InstanceIDModel attribute), 31
objects (morango.models.MorangoMPTTModel attribute), 29
objects (morango.models.Nonce attribute), 27
objects (morango.models.RecordMaxCounter attribute), 40
objects (morango.models.RecordMaxCounterBuffer attribute), 40
objects (morango.models.ScopeDefinition attribute), 28
objects (morango.models.SharedKey attribute), 23
objects (morango.models.Store attribute), 37
objects (morango.models.SyncableModel attribute), 41
objects (morango.models.SyncSession attribute), 33
objects (morango.models.TransferSession attribute), 35

parent (morango.models.Certificate attribute), 24
parent_id (morango.models.Certificate attribute), 25
partition (morango.models.DatabaseMaxCounter attribute), 39
platform (morango.models.InstanceIDModel attribute), 31
prepare_request() (morango.sync.syncsession.SessionWrapper method), 42
prepare_value() (morango.models.SyncSession attribute), 33
prepare_value() (morango.models.UUIDField method), 22
primary_scope_param_key (morango.models.ScopeDefinition attribute), 28
private_key (morango.models.Certificate attribute), 25
private_key (morango.models.SharedKey attribute), 23
PublicKeyField (class in morango.models), 22
profile (morango.models.Certificate attribute), 25
profile (morango.models.DeletedModels attribute), 36
profile (morango.models.HardDeletedModels attribute), 37
profile (morango.models.ScopeDefinition attribute), 28
profile (morango.models.SyncSession attribute), 33
public_key (morango.models.Certificate attribute), 25
public_key (morango.models.SharedKey attribute), 23
PullClient (class in morango.sync.syncsession), 44
push (morango.models.TransferSession attribute), 35
push_signed_client_certificate_chain() (morango.sync.syncsession.NetworkSyncConnection method), 43
PushClient (class in morango.sync.syncsession), 44

queuing (morango.sync.syncsession.SyncClientSignals attribute), 44

read_filter_template (morango.models.ScopeDefinition attribute), 28
read_write_filter_template (morango.models.ScopeDefinition attribute), 28
RecordMaxCounter (class in morango.models), 39
RecordMaxCounter.DoesNotExist, 40
RecordMaxCounter.MultipleObjectsReturned, 40
recordmaxcounter_set (morango.models.Store attribute), 37
RecordMaxCounterBuffer (class in morango.models), 40
RecordMaxCounterBuffer.DoesNotExist, 40
RecordMaxCounterBuffer.MultipleObjectsReturned, 40
recordmaxcounterbuffer_set
(morango.models.TransferSession attribute), 35
records_total (morango.models.TransferSession attribute), 35
records_transferred (morango.models.TransferSession attribute), 35
request() (morango.sync.session.SessionWrapper method), 42
reset_transfer_bytes() (morango.sync.session.SessionWrapper method), 43
retrieve_by_id() (morango.models.ScopeDefinition class method), 28

salt (morango.models.Certificate attribute), 25
save() (morango.models.DatabaseIDModel method), 30
save() (morango.models.SyncableModel method), 42
save_certificate_chain() (morango.models.Certificate class method), 25
Scope (class in morango.models), 28
scope_definition (morango.models.Certificate attribute), 25
scope_definition_id (morango.models.Certificate attribute), 25
scope_params (morango.models.Certificate attribute), 25
scope_version (morango.models.Certificate attribute), 25
ScopeDefinition (class in morango.models), 27
ScopeDefinition.DoesNotExist, 27
ScopeDefinition.MultipleObjectsReturned, 27
ScopeDefinition.MultipleObjectsReturned, 27
serialize() (morango.models.Certificate method), 25
serialize() (morango.models.SyncableModel method), 42
serialized (morango.models.Certificate attribute), 25
server_certificate (morango.models.SyncSession attribute), 33
server_certificate_id (morango.models.SyncSession attribute), 33
server_fsic (morango.models.TransferSession attribute), 35
server_instance (morango.models.SyncSession attribute), 33
server_ip (morango.models.SyncSession attribute), 33
SessionWrapper (class in morango.sync.session), 42
SharedKey (class in morango.models), 22
SharedKey.DoesNotExist, 22
SharedKey.MultipleObjectsReturned, 22
sign() (morango.models.Certificate method), 25
sign_certificate() (morango.models.Certificate method), 25
signals (morango.sync.syncsession.BaseSyncClient attribute), 43
signals (morango.sync.syncsession.SyncSessionClient attribute), 45
signature (morango.models.Certificate attribute), 25
start_timestamp (morango.models.TransferSession attribute), 35
start_timestamp (morango.models.TransferSession attribute), 35
Store (class in morango.models), 37
Store.DoesNotExist, 37
Store.MultipleObjectsReturned, 37
store_model (morango.models.RecordMaxCounter attribute), 40
store_model_id (morango.models.RecordMaxCounter attribute), 40
sync_filter (morango.sync.syncsession.PullClient attribute), 44
sync_session (morango.models.TransferSession attribute), 36
sync_session_id (morango.models.TransferSession attribute), 36
SyncableModel (class in morango.models), 41
SyncableModel.Meta (class in morango.models), 41
SyncableModelManager (class in morango.models), 28
SyncableModelQuerySet (class in morango.models), 28
SyncSession (class in morango.models), 31
SyncSession.DoesNotExist, 32
SyncSession.MultipleObjectsReturned, 32
SyncSessionClient (class in morango.sync.syncsession), 44
SyncSessionPermissions (class in morango.api.permissions), 45
synchronsions_client (morango.models.Certificate attribute), 26
synchronsions_server (morango.models.Certificate attribute), 26
system_id (morango.models.InstanceIDModel attribute), 31
sysversion (morango.models.InstanceIDModel attribute), 31

timestamp (morango.models.Nonce attribute), 27
to_python() (morango.models.PrivateKeyField method), 22

Index 53
morango Documentation

to_python() (morango.models.PublicKeyField method), 22
to_python() (morango.models.UUIDField method), 22
transfer_session (morango.models.Buffer attribute), 38
transfer_session (morango.models.RecordMaxCounterBuffer attribute), 40
transfer_session_id (morango.models.Buffer attribute), 39
transfer_session_id (morango.models.RecordMaxCounterBuffer attribute), 41
transferring (morango.sync.syncsession.SyncClientSignals attribute), 44
TransferSession (class in morango.models), 34
TransferSession.DoesNotExist, 34
TransferSession.MultipleObjectsReturned, 34
transfersession_set (morango.models.SyncSession attribute), 33
TransferSessionPermissions (class in morango.api.permissions), 45
tree_id (morango.models.Certificate attribute), 26

U
update() (morango.models.SyncableModelQuerySet method), 28
update_fsics() (morango.models.DatabaseMaxCounter class method), 39
urlresolve() (morango.sync.syncsession.NetworkSyncConnection method), 43
use_nonce() (morango.models.Nonce class method), 27
uuid_input_fields (morango.models.Certificate attribute), 26
uuid_input_fields (morango.models.DatabaseIDModel attribute), 30
uuid_input_fields (morango.models.InstanceIDModel attribute), 31
uuid_input_fields (morango.models.Nonce attribute), 27
uuid_input_fields (morango.models.UUIDModelMixin attribute), 23
UUIDField (class in morango.models), 21
UUIDModelMixin (class in morango.models), 23
UUIDModelMixin.Meta (class in morango.models), 23

V
verify() (morango.models.Certificate method), 26
version (morango.models.ScopeDefinition attribute), 28

W
write_filter_template (morango.models.ScopeDefinition attribute), 28