Technical University of Košice
Faculty of Electrical Engineering and Informatics

Atmosphere: Concurrency enabled data synchronization platform with HTML5/JS and Cocoa clients

Appendix B: System guide

2012 **Vojtech Riník**

Technical University of Košice

Faculty of Electrical Engineering and Informatics

Atmosphere: Concurrency enabled data synchronization platform with HTML5/JS and Cocoa clients

Appendix B: System guide

Study Programme:	Informatics
Field of study:	9.2.1 Informatics
Department:	Department of Computers and Informatics (DCI)
Supervisor:	Assoc. Prof. Ing. František Jakab, PhD.
Consultant(s):	Ing. Ivan Klimek

Košice 2012 **Vojtech Riník**

B System guide

B.1 JavaScript library

All of the following sources are available in the GitHub repository of Atmosphere.

```
Spine     = require('spine')
SocketIO  = require('./vendor/socket.io')
window.SocketIO = SocketIO

MessageClient   = require('./message_client')
AppContext      = require('./app_context')
MetaContext     = require('./meta_context')
ResourceClient  = require('./resource_client')

# Atmosphere.Synchronizer
#
# The main interface used mostly for configuration and management of
   the
# synchronization.
# ------------------------------------------------------------------

class Synchronizer extends Spine.Module
  @include Spine.Events

  # Object lifecycle
  # ----------------------------------------------------------------

  constructor: (options) ->
    @messageClient = new MessageClient(this)
    @metaContext = new MetaContext()
    @appContext = new AppContext()
    @resourceClient = new ResourceClient(sync: this, appContext:
       @appContext)
    @_needsSync = false
    @_isSyncInProgress = false
```

```
  Synchronizer.instance = this
  Synchronizer.res = @resourceClient

# Resource interface
# ------------------------------------------------------------------

updateOrCreate: (uri, item) ->
  # Check for ID change
  if item.id && item.id != uri.id
    console.log "changing id #{uri.id} -> #{item.id}"
    @appContext.changeID(uri, item.id)
    @metaContext.changeIDAtURI(uri, item.id)
    uri.id = item.id
  @appContext.updateOrCreate(uri, item)

# Resource interface
# ------------------------------------------------------------------

fetch: (params...) ->
  @resourceClient.fetch(params...)

save: (object, options) ->
  if options.sync
    object.save()
    uri = @appContext.objectURI(object)
    options = object.remoteSaveOptions(options) if object.
        remoteSaveOptions?
    @resourceClient.save(object, options)
  else
    object.save()
    @markObjectChanged(object)
    @setNeedsSync()

execute: (params...) -> @resourceClient.execute(params...)
request: (params...) -> @resourceClient.request(params...)
```

```
# Meta objects
# ------------------------------------------------------------

markObjectChanged: (object) ->
  uri = @appContext.objectURI(object)
  @metaContext.markURIChanged(uri)

markURISynced: (uri) ->
  @metaContext.markURISynced(uri)

# Synchronization
# ------------------------------------------------------------

setNeedsSync: ->
  @_needsSync = true
  @startSync()

startSync: ->
  return unless @_needsSync == true
  # return if @_isSyncInProgress == true
  @_isSyncInProgress = true
  resourceClient = @resourceClient
  @metaContext.changedObjects (metaObjects) =>
    for metaObject in metaObjects
      action = if metaObject.isLocalOnly then "create" else "update"
      console.log "syncing meta object #{action}", metaObject
      object = @appContext.objectAtURI(metaObject.uri)
      options = {action: action}
      options = object.remoteSaveOptions(options) if object.
          remoteSaveOptions?
      resourceClient.save(object, options)
      # TODO: Finish sync
```

```
  removeObjectsNotInList: (collection, ids, scope) ->
    uris = @appContext.allURIs(collection, scope)
    for uri in uris
      isInList = ids.indexOf(uri.id) != -1
      if !isInList
        @metaContext.isURILocalOnly uri, (res) =>
          return if res == true # Don't destroy if object is local only
          console.log "[ResourceClient] Local id #{uri.id} wasn't
              retrieved, destroying."
          @appContext.destroy(uri)

  # Auth
  # ------------------------------------------------------------------

  setAuthKey: (key) ->
    @authKey = key

  hasAuthKey: ->
    @authKey? && @authKey != ""

  didAuth: (content) ->
    @trigger("auth_success")
    @getChanges()

  didFailAuth: (content) ->
    @trigger("auth_fail")


module.exports = Synchronizer
```

Listing 1: synchronizer.coffee

```
String.prototype.underscorize = ->
        @replace /([A-Z])/g, (letter) -> "_#{letter.toLowerCase()}".
            substr(1)
```

```
class AppContext
  constructor: ->
    @_models = {}

  exists: (uri) ->
    model = @_modelForURI(uri)
    !!model.exists(uri.id)

  updateOrCreate: (uri, data) ->
    if @exists(uri)
      @update uri, data
    else
      @create uri, data

  create: (uri, data) ->
    model = @_modelForURI(uri)
    record = new model(data)
    record.id = uri.id if uri.id?
    record.save()
    uri.id = record.id
    model.fetch()
    record

  update: (uri, data) ->
    record = @objectAtURI(uri)
    record.updateAttributes(data)
    record

  changeID: (uri, id) ->
    record = @objectAtURI(uri)
    console.log "changing id from #{record.id} to #{id}"
    record.changeID(id)

  relation: (name, sourceURI, targetURI) ->
```

```
    source = @objectAtURI(sourceURI)
    target = @objectAtURI(targetURI)
    hash = {}
    hash[name] = target
    source.updateAttributes(hash)
    source.save()

  objectAtURI: (uri) ->
    model = @_modelForURI(uri)
    model.find(uri.id)

  dataForURI: (uri) ->
    @dataForObject(@objectAtURI(uri))

  dataForObject: (object) ->
    object.attributes()

  _modelForURI: (uri) ->
    model = @_models[uri.collection]
    unless model
      console.log "Initializing model", uri.collection
      model = require("models/#{uri.collection.underscorize()}")
      model.fetch()
      @_models[uri.collection] = model
    model

  objectURI: (object) ->
    {collection: object.constructor.className, id: object.id}

  allURIs: (collection, predicate) ->
    uri = {collection:collection}
    model = @_modelForURI(uri)
    # model.fetch() # TODO: Fetch maybe?
    objects = if predicate?
      model.select(predicate)
```

```
    else
      model.all()
    @objectURI(object) for object in objects

  destroy: (uri) ->
    @objectAtURI(uri).destroy()

module.exports = AppContext
```

Listing 2: $\mathbf{app}_{c}ontext.coffee$

```
Lawnchair = require('./vendor/lawnchair')

KeyFromURI = (uri) ->
  "#{uri.collection}.#{uri.id}"

URIFromKey = (key) ->
  [collection, id] = key.split(".")
  {collection: collection, id: id}



# Atmosphere.MetaContext
#
# This class manages "meta" objects. Every application object has a
   meta object
# where all synchronization-related information is stored.
#
   ================================================================================



class MetaContext
  constructor: ->
    @configure()

  configure: ->
```

```
    # console.log "configuring"
    new Lawnchair {db: "atmosphere", name: "Meta", adapter: window.
        LawnchairAdapter}, (store) =>
      @store = store

  # Marking changes
  # ----------------------------------------------------------------

  # Marks object at URI as changed.
  markURIChanged: (uri) ->
    @findOrCreateObjectAtURI uri, (object) =>
      object.isChanged = true
      @saveObject(object)

  findOrCreateObjectAtURI: (uri, callback) ->
    @objectAtURI uri, (object) =>
      if object then callback(object) else @createObjectAtURI(uri,
          callback)

  objectAtURI: (uri, callback) ->
    @store.get KeyFromURI(uri), (dict) ->
      if dict? then callback(new MetaObject(dict)) else callback(null)

  createObjectAtURI: (uri, callback) ->
    object = {key: KeyFromURI(uri), isChanged: false, isLocalOnly: true
        }
    @store.save object, ->
      # console.log "creating meta object for", object
      callback(new MetaObject(object))

  saveObject: (object) ->
    # console.log "saving", object, object.storeDict()
    @store.save(object.storeDict())

  deleteObject: (object) ->
```

```
    @store.remove object.storeKey(), ->

  changeIDAtURI: (uri, id) ->
    @objectAtURI uri, (object) =>
      return unless object
      @deleteObject(object)
      object.uri.id = id
      @saveObject(object)

  # Getting changed objects
  # ----------------------------------------------------------------

  isURILocalOnly: (uri, callback) ->
    @objectAtURI uri, (object) ->
      return callback(true) unless object
      callback(object.isLocalOnly)

  isURIChanged: (uri, callback) ->
    @objectAtURI uri, (object) ->
      return callback(false) unless object
      callback(object.isChanged)

  changedObjects: (callback) ->
    changed = []
    @store.all (dicts) ->
      for dict in dicts
        object = new MetaObject(dict)
        changed.push(object) if object.isChanged == true
      callback(changed)

  # Marking local/remote
  # ----------------------------------------------------------------

  markURISynced: (uri) ->
    @findOrCreateObjectAtURI uri, (object) =>
```

```
      object.isLocalOnly  = false
      object.isChanged    = false
      @saveObject(object)

# Atmosphere.MetaObject
#
# Represents a meta object.
#
   ================================================================================


class MetaObject
  constructor: (attrs) ->
    return null unless attrs.key
    @uri = URIFromKey(attrs.key)
    @isChanged = attrs.isChanged
    @isLocalOnly = attrs.isLocalOnly

  storeDict: ->
    {key: @storeKey(), isChanged: @isChanged, isLocalOnly: @isLocalOnly
       }

  storeKey: ->
    KeyFromURI(@uri)


module.exports = MetaContext
```

Listing 3: meta$_{c}ontext.coffee$

```
Spine = require('spine')
{assert} = require('./util')

class ResourceClient
  constructor: (options) ->
```

```
    @sync = options.sync
    @appContext = options.appContext

    @base = null
    @headers = {}
    @routes = null
    @IDField = "id"
    @dataCoding = "form" # "json"
    @subitems = {}

  fetch: (model, options = {}) ->
    collection = model.className
    path = @_findPath(collection, "index", options)
    ids = []
    @request path, {}, (result) =>
      items = @itemsFromResult(result)
      unless items?
        console.log "[ResourceClient] Items not found in response",
            result
        return
      ids = @updateFromItems(collection, items, options)
      @_removeObjectsNotInList(collection, ids, options.removeScope) if
          options.remove == true
      options.success() if options.success

  updateFromItems: (collection, items, options) ->
    ids = []
    for item in items
      uri = {collection: collection}
      object = @updateFromItem(uri, item, options)
      ids.push(object.id)
    ids

  updateFromItem: (uri, item, options = {}) ->
    item.id = item[@IDField]
```

```
    assert item.id, "[ResourceClient] There's no field '#{@IDField}'
       that is configured as IDField in incoming object"
    uri.id or= item.id
    options.updateData(item) if options.updateData?
    if options.updateFromData?
      options.updateFromData(uri, item, @_updateFromData)
    else
      @_updateFromData(uri, item)

  _updateFromData: (uri, data) =>
    object = @sync.updateOrCreate(uri, data)
    @sync.markURISynced(uri)
    object

  _removeObjectsNotInList: (collection, ids, scope) ->
    @sync.removeObjectsNotInList(collection, ids, scope)

  itemsFromResult: (result) ->
    result

  save: (object, options = {}) ->
    uri = @appContext.objectURI(object)
    path = @_findPathForURI(uri, options.action, options)
    data = options.data || @appContext.dataForObject(object)
    data[@IDField] = object.id unless data[@IDField]?
    data = options.prepareData(data, options) if options.prepareData?
    @request path, data, (result) =>
      if options.sync
        object.save()
        uri = @appContext.objectURI(object)
      @updateFromItem(uri, result, options)

  execute: (options, callback) ->
    if typeof options == 'string'
      path = {method: 'get', path: options}
```

```
    else if options.collection
      path = @_findPath(options.collection, options.action, options)
    else if options.object
      path = @_findPathForObject(options.object, options.action,
          options)
    else
      path = options
    @request path, options.data, callback

  _findPathForObject: (object, action, options) ->
    uri = @appContext.objectURI(object)
    @_findPathForURI(uri)

  _findPathForURI: (uri, action, options) ->
    options.pathParams    or= {}
    options.pathParams.id or= uri.id
    @_findPath(uri.collection, options.action, options)

  _findPath: (collection, action, options = {}) ->
    assert @routes[collection], "No route found for #{collection}"
    path = @routes[collection][action]
    assert path, "No route found for #{collection}/#{action}"
    [method, path] = path.split(" ")
    if options.pathParams?
      path = path.replace(":#{param}", value) for param, value of
          options.pathParams
    route = {method: method, path: path}
    route.query = $.param(options.params) if options.params?
    route

  request: (path, data, callback) ->
    proceed = =>
      contentType = "application/x-www-form-urlencoded"
      if @dataCoding == "json"
        data = JSON.stringify(data)
```

```
        contentType = "application/json"
      success = (result) ->
        callback(result) if callback
      error = (res, err) =>
        if res.status == 401
          console.log "failed with error 401 #{err}"
          return @sync.didFailAuth()
        console.log "Request failed #{res} #{err}", res, err
      options =
        dataType: "json"
        success: success
        error: error
        headers: @headers
        contentType: contentType
      options.data = data if data?
      @ajax path, options
    if @beforeRequest?
      @beforeRequest(proceed)
    else
      proceed()

  ajax: (path, options = {}) ->
    path = {path: path} if typeof path == 'string'
    url = @base + path.path
    url += "?#{path.query}" if path.query
    options.type or= path.method
    $.ajax url, options


  addHeader: (header, value) ->
    @headers[header] = value

module.exports = ResourceClient
```

Listing 4: resource_client.coffee

```
# Atmosphere.Client
#
# This class is responsible for socket communication through messages,
   thus
# the name, "Message" client.
# ------------------------------------------------------------------

class MessageClient
  constructor: (sync) ->
    @sync = sync

  connect: (callback) ->
    @close()
    console.log "[Atmosphere.Client] Connecting to #{@url}"
    @socket = SocketIO.connect(@url, 'force new connection': true)
    @socket.on 'connect', =>
      console.log 'socket connected'
      @send 'auth', @authKey
    @socket.on 'notification', this.parseNotification
    @socket.on 'update', this.parseUpdate
    @socket.on 'disconnect', this.socketDidClose

  close: ->
    @socket.disconnect() if @socket

  socketDidClose: =>
    console.log "[Atmosphere.Client] Connection closed"
    @socket = null

  send: (type, content) ->
    @socket.emit(type, content)

  # Messaging interface
  # ----------------------------------------------------------------
```

```
  parseNotification: (data) =>
    console.log 'notification: ', data

  parseUpdate: (data) =>
    @sync.updateOrCreate(data.uri, data.attrs)

module.exports = MessageClient
```

Listing 5: message_client.coffee

```
# Atmosphere Spine Model Adapter
# ----------------------------------------------------------------

Spine       = require('spine')
Atmosphere  = require('./synchronizer')
require('./lawnchair_spine')

Spine.Model.Atmosphere =
  extended: ->
    @extend Spine.Model.Lawnchair
    spineSave = @::["save"]
    @::["save"] = (args...) ->
      atmos = Atmosphere.instance
      options = args[0]
      if atmos? && options? && options.remote == true
        atmos.save(this, options)
      else
        spineSave.call(this, args...)
    @::["changeID"] = (id) -> # TODO: Fix this mess
      @destroy()
      @id = id
      @newRecord = true
      @save()
    @bind 'beforeCreate', (record) ->
      record.id or= @_uuid()
```

```
_uuid: ->
  'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace /[xy]/g, (c) ->
    r = Math.random() * 16 | 0
    v = if c is 'x' then r else r & 3 | 8
    v.toString 16

# uid: -> @_uuid()

sync: (params = {}) ->
  @fetch()
  atmos = Atmosphere.instance
  if atmos? && params.remote == true
    atmos.fetch(@, params)
```

Listing 6: spine.coffee

B.2 Cocoa library

The source code for Cocoa library is not included here due to its size. It can be found in GitHub repository of Atmosphere.

Here follows the documentation for main classes and data structures of Cocoa implementation.

B.2.1 _ATObjectURI Struct Reference

Public Attributes

NSString * **entity**

NSString * **identifier**

The documentation for this struct was generated from the following file:

ATObjectURI.h

B.2.2 _ATRoute Struct Reference

Public Attributes

RKRequestMethod **method**

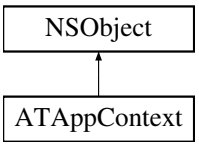
NSString * **path**

The documentation for this struct was generated from the following file:

ATResourceClient.h

B.2.3 ATAppContext Class Reference

Inheritance diagram for ATAppContext:



Public Member Functions

(id) - **initWithSynchronizer:appContext:**

(NSManagedObject *) - **objectAtURI:**

(NSManagedObject *) - **createAppObjectAtURI:**

(Class) - **_managedClassForURI:**

(ATObjectURI) - **URIOfAppObject:**

(void) - **changeIDTo:atURI:**

(void) - **updateAppObject:withDictionary:**

(void) - **deleteAppObject:**

(void) - **_resolveRelations:withDictionary:**

(NSDictionary *) - **dataForObject:**

(NSArray ∗) - **relationsForAppObject:**

(BOOL) - **attributesChangedInAppObject:**

(BOOL) - **hasChanges**

(void) - **save**

(void) - **save:**

(void) - **obtainPermanentIDsForObjects:error:**

Static Public Member Functions

(id) + **sharedAppContext**

Protected Attributes

ATSynchronizer ∗ **_sync**

NSManagedObjectContext ∗ **_managedContext**

NSMutableArray ∗ **_relationsQueue**

Properties

ATSynchronizer ∗ **sync**

NSManagedObjectContext ∗ **managedContext**

ATAttributeMapper ∗ **attributeMapper**

The documentation for this class was generated from the following files:

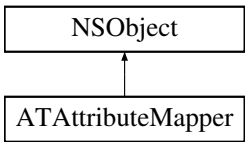
ATAppContext.h

ATAppContext.m

B.2.4 ATAttributeMapper Class Reference

```
#import <ATAttributeMapper.h>
```

Inheritance diagram for ATAttributeMapper:



Public Member Functions

(id) - **initWithMappingHelper:**

Properties

ATMappingHelper ∗ **mappingHelper**

B.2.5 Detailed Description

This class does nothing

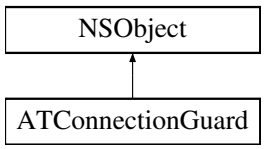
The documentation for this class was generated from the following files:

ATAttributeMapper.h

ATAttributeMapper.m

B.2.6 ATConnectionGuard Class Reference

Inheritance diagram for ATConnectionGuard:



Public Member Functions

(void) - **start**

(void) - **stop**

(void) - **_checkConnection**

Protected Attributes

BOOL **isRunning**

Properties

ATSynchronizer * **client**

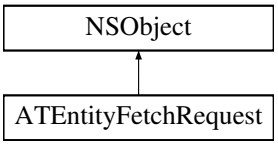
The documentation for this class was generated from the following files:

ATConnectionGuard.h

ATConnectionGuard.m

B.2.7 ATEntityFetchRequest Class Reference

Inheritance diagram for ATEntityFetchRequest:



Public Member Functions

(id) - **initWithResourceClient:entity:**

(void) - **send**

(ATObjectURI) - **objectURIFromItem:**

Properties

ATResourceClient * **resourceClient**

RKClient * **networkClient**

NSString * **entity**

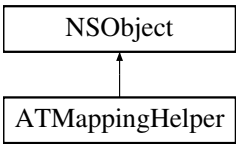
The documentation for this class was generated from the following files:

ATEntityFetchRequest.h

ATEntityFetchRequest.m

B.2.8 ATMappingHelper Class Reference

Inheritance diagram for ATMappingHelper:



Public Member Functions

(void) - **loadEntitiesMapFromResource:**

(void) - **loadAttributesMapFromResource:**

(void) - **loadRelationsMapFromResource:**

(void) - **_loadResource:intoDictionary:**

(NSString ∗) - **localEntityNameFor:**

(NSString ∗) - **serverEntityNameFor:**

(NSString ∗) - **serverEntityNameForAppObject:**

(NSString ∗) - **serverAttributeNameFor:entity:**

(NSString ∗) - **localAttributeNameFor:entity:**

(NSDictionary ∗) - **relationsForObject:**

(NSDictionary ∗) - **relationsForEntity:**

Properties

NSDictionary ∗ entitiesMap

NSDictionary ∗ **attributesMap**

NSDictionary ∗ **relationsMap**

B.2.9 Property Documentation

B.2.10 - (NSDictionary∗) entitiesMap `[read, write, retain]`

Entities map dictionary. Key represents server entity name and value represents client entity name

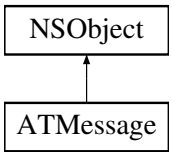
The documentation for this class was generated from the following files:

ATMappingHelper.h

ATMappingHelper.m

B.2.11 ATMessage Class Reference

Inheritance diagram for ATMessage:



Public Member Functions

(NSString ∗) - **JSONString**

Static Public Member Functions

(ATMessage ∗) + **messageFromJSONString:**

Protected Attributes

NSString ∗ **_type**

NSDictionary ∗ **_content**

Properties

NSString * **type**

NSDictionary * **content**

The documentation for this class was generated from the following files:

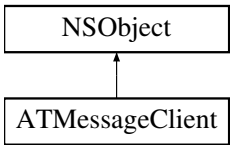
ATMessage.h

ATMessage.m

B.2.12 ATMessageClient Class Reference

`#import <ATMessageClient.h>`

Inheritance diagram for ATMessageClient:



Public Member Functions

(id) - **initWithSynchronizer:**

(void) - **connect**

(BOOL) - **isConnected**

(void) - **_initializeSocketConnection**

(void) - **_sendConnectMessage**

(void) - **disconnect**

(void) - **_didReceiveServerAuthFailure:**

(void) - **_didReceiveServerAuthSuccess:**

(void) - **sendMessage:**

(void) - **_didReceiveServerPush:**

Protected Attributes

BOOL **_isRunning**

ATSynchronizer * **_sync**

NSString * _host

NSInteger **_port**

SocketIO * **_connection**

Properties

ATSynchronizer * **sync**

NSString * **host**

NSInteger **port**

SocketIO * **connection**

B.2.13 Detailed Description

ATMessageClient is responsible for dealing with live connection for push updates and notifications.

B.2.14 Member Data Documentation

B.2.15 - (NSString*) _host [protected]

Connection

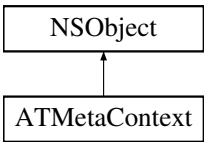
The documentation for this class was generated from the following files:

ATMessageClient.h

ATMessageClient.m

B.2.16 ATMetaContext Class Reference

Inheritance diagram for ATMetaContext:



Public Member Functions

(BOOL) - **save**

(void) - **markURIChanged:**

(void) - **markURISynced:**

(ATMetaObject ∗) - **objectAtURI:**

(ATMetaObject ∗) - **ensureObjectAtURI:**

(ATMetaObject ∗) - **createObjectAtURI:**

(NSArray ∗) - **changedObjects**

(void) - **changeIDTo:atURI:**

Static Public Member Functions

(id) + **restore**

(NSString ∗) + **path**

Protected Attributes

NSMutableDictionary ∗ **_objects**

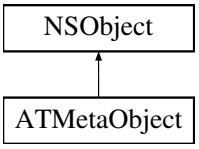
The documentation for this class was generated from the following files:

ATMetaContext.h

ATMetaContext.m

B.2.17 ATMetaObject Class Reference

Inheritance diagram for ATMetaObject:



Public Member Functions

(id) - **initWithURI:**

Properties

ATObjectURI **uri**

BOOL **isChanged**

BOOL **isLocalOnly**

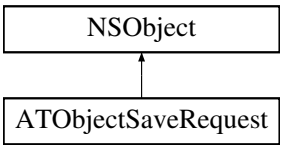
The documentation for this class was generated from the following files:

ATMetaObject.h

ATMetaObject.m

B.2.18 ATObjectSaveRequest Class Reference

Inheritance diagram for ATObjectSaveRequest:



Public Member Functions

(id) - **initWithResourceClient:object:options:**

(void) - **send**

Properties

ATResourceClient * **resourceClient**

RKClient * **networkClient**

NSDictionary * **options**

NSManagedObject * **object**

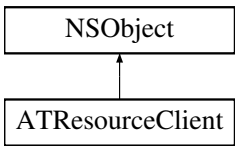
The documentation for this class was generated from the following files:

ATObjectSaveRequest.h

ATObjectSaveRequest.m

B.2.19 ATResourceClient Class Reference

Inheritance diagram for ATResourceClient:



Public Member Functions

(id) - **initWithSynchronizer:**

(void) - **setBaseURL:**

(void) - **addHeader:withValue:**

(void) - **fetchEntity:**

(void) - **didFetchItem:withURI:**

(void) - **saveObject:**

(void) - **saveObject:options:**

(void) - **loadRoute:params:delegate:**

(void) - **loadRoutesFromResource:**

(ATRoute) - **routeForEntity:action:**

(ATRoute) - **routeForEntity:action:params:**

Properties

ATSynchronizer * **sync**

RKClient * **client**

NSDictionary * **routes**

NSString * **IDField**

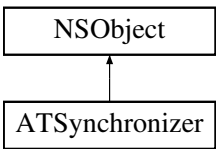
The documentation for this class was generated from the following files:

ATResourceClient.h

ATResourceClient.m

B.2.20 ATSynchronizer Class Reference

Inheritance diagram for ATSynchronizer:



Public Member Functions

(id) - **initWithAppContext:**

(void) - **close**

(NSString *) - **authKeyOrNull**

(void) - **fetchEntity:**

(void) - **syncObject:**

(void) - **startSync**

(void) - **sync**

(void) - **updateObjectAtURI:withDictionary:**

(void) - **changeURIFrom:to:**

(void) - **startAutosync**

(void) - **stopAutosync**

(void) - **_didChangeAppObject:**

Protected Attributes

ATMappingHelper * _mappingHelper

ATMetaContext * _metaContext

ATAppContext * **_appContext**

ATMessageClient * _messageClient

ATResourceClient * **_resourceClient**

NSString * _authKey

BOOL **_isSyncScheduled**

Properties

ATMetaContext * **metaContext**

ATAppContext * **appContext**

ATMappingHelper * **mappingHelper**

ATMessageClient * **messageClient**

ATResourceClient * **resourceClient**

NSString * **authKey**

id< ATSynchronizerDelegate > delegate

B.2.21 Member Data Documentation

B.2.22 - (NSString∗) _authKey [protected]

State

B.2.23 - (ATMappingHelper∗) _mappingHelper [protected]

Helper

B.2.24 - (ATMessageClient∗) _messageClient [protected]

Networking clients

B.2.25 - (ATMetaContext∗) _metaContext [protected]

Context

B.2.26 Property Documentation

B.2.27 - (id< ATSynchronizerDelegate >) delegate [read, write, assign]

Delegate

The documentation for this class was generated from the following files:

ATSynchronizer.h

ATSynchronizer.m

B.2.28 <ATSynchronizerDelegate> Protocol Reference

Public Member Functions

(void) - **clientAuthDidSucceed:**

(void) - **clientAuthDidFail:**