Caching API

Backendless Caching API provides a way to temporarily store data on the server in a highly efficient in-memory cache. The cache storage accepts key-value pairs where key is a string and value can be a primitive or complex data structure (strings, numbers, arrays, dictionaries, complex types, etc). The caching mechanism is cross-platform, that means Backendless automatically takes care of adapting data between heterogeneous client types. For instance, an Android client can put into cache an instance of Java object of type Person and an iOS (or any other) client can retrieve that object as an instance of the corresponding iOS (or other client type) class.

Restrictions

- Maximum time duration a value can stay in cache is 2 hours (7200 seconds), but it's "life" can be extended with an API request.
- Before an object is placed in cache, it is serialized as a byte array. The size of the serialized object cannot exceed 10240 bytes.
- Customers on the Free plan can store up to 50 objects in cache (concurrent storage), Backendless Plus - 200 objects, Cloud Enterprise - 1000 objects.

The Caching API supports the following functions:

Putting data into cache

This API request places the object into Backendless cache and maps it to the specified key. If the `timeToLive` argument is not set, the object will expire from cache in 2 hours from the time when it is put in cache. All methods are available via `backendless.cache` accessor:

```
// sync methods with fault option
-(BOOL)put:(NSString *)key object:(id)entity fault:(Fault **)fault;
-(BOOL)put:(NSString *)key object:(id)entity timeToLive:(int)seconds fault:(Fault
**)fault;

// async methods with responder
-(void)put:(NSString *)key object:(id)entity responder:(id<IResponder>)responder;
-(void)put:(NSString *)key object:(id)entity timeToLive:(int)seconds
responder:(id<IResponder>)responder;

// async methods with block-based callback
-(void)put:(NSString *)key object:(id)entity response:(void (^)(id))responseBlock
error:(void (^)(Fault *))errorBlock;
-(void)put:(NSString *)key object:(id)entity timeToLive:(int)seconds response:(void
```

```
(^)(id))responseBlock error:(void (^)(Fault *))errorBlock;
```

where:

`key`	- key assigned to the object to identify it in cache. The key is used to retrieve the object from cache or to check if the cache still contains the object.
`entity`	- object to place into cache.
`timeToLive`	- numeric value (in seconds) indicating how long the object must stay in cache before it is expires. When an object expires, Backendless automatically removes it from cache. The default value is 7200 seconds.
`responder, response, error`	- the callbacks used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Examples:

```
// sync

Weather *entity = [Weather new];
Fault *fault = nil;
[backendless.cache put:@"CacheWeather" object:entity fault:&fault];
if (fault) {
    NSLog(@"FAULT: %@", fault);
    return;
}

NSLog(@"[SYNC] object has been placed into cache: %@", entity);

// async

Weather *entity = [Weather new];
[backendless.cache put:@"CacheWeather" object:entity  timeToLive:1800
response:^(id result) {
        NSLog(@"[ASYNC] object has been placed into cache: %@", entity);
 }
 error:^(Fault *fault) {
        NSLog(@"FAULT: %@", fault);
 }];
```

Retrieving data from cache

This API request retrieves an object from Backendless cache. If object is not present in cache, the method returns null for complex

types or default value for primitive values. All methods are available via `backendless.cache` accessor:

```
// sync methods with fault option
-(id)get:(NSString *)key fault:(Fault **)fault;

// async methods with responder
-(void)get:(NSString *)key responder:(id<IResponder>)responder;

// async methods with block-based callback
-(void)put:(NSString *)key object:(id)entity response:(void (^)(id))responseBlock -
(void)get:(NSString *)key response:(void (^)(id))responseBlock error:(void (^)(Fault
*))errorBlock;
```

where:

`key`	- key assigned to the object to identify it in cache. The key is used to retrieve the object from cache or to check if the cache still contains the object.
`responder, response, error`	- the callbacks used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Examples:

```
// sync

Fault *fault = nil;
Weather *entity = [backendless.cache get:@"CacheWeather" fault:&fault];
if (fault) {
    NSLog(@"FAULT: %@", fault);
    return;
}

NSLog(@"[SYNC] retrieved order from cache: %@", entity);

// async

[backendless.cache get:@"CacheWeather"
response:^(id entity) {
        NSLog(@"[ASYNC] retrieved order from cache: %@", entity)
 }
 error:^(Fault *fault) {
        NSLog(@"FAULT: %@", fault);
 }];
```

Checking if key exists in cache

This API request checks if an object exists in cache. If object is present in cache, the method returns true, otherwise - false. All methods are available via `backendless.cache` accessor:

```
// sync methods with fault option
-(NSNumber *)contains:(NSString *)key fault:(Fault **)fault;

// async methods with responder
-(void)contains:(NSString *)key responder:(id<IResponder>)responder;

// async methods with block-based callback
-(void)contains:(NSString *)key response:(void (^)(NSNumber *))responseBlock
error:(void (^)(Fault *))errorBlock;
```

where:

`key`	- key assigned to the object to identify it in cache. The key is used to retrieve the object from cache or to check if the cache still contains the object.
`responder, response, error`	- the callbacks used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Examples:

```
// sync

Fault *fault = nil;
NSNumber *result = [backendless.cache contains:@"CacheWeather" fault:&fault];
if (fault) {
    NSLog(@"FAULT: %@", fault);
    return;
}

NSLog(@"[SYNC] object exists in cache: %@", [result boolValue]?@”YES”:@”NO”);

// async

[backendless.cache contains:@"CacheWeather"
response:^( NSNumber *result) {
        NSLog(@"[ASYNC] object exists in cache: %@", [result
        boolValue]?@”YES”:@”NO”); }
 error:^(Fault *fault) {
        NSLog(@"FAULT: %@", fault);
 }];
```

Extending object's life in cache

There are two way to extend object's life in cache - relative timeframe and fixed timeframe. With the relative timeframe a period of time is added to the timestamp of the call to determine the new expiration time. The fixed timestamp approach sets the timestamp when the object must expire from cache. All methods are available via `backendless.cache` accessor:

```
// sync methods with fault option
-(BOOL)expireIn:(NSString *)key timeToLive:(int)seconds fault:(Fault **)fault;
-(BOOL)expireAt:(NSString *)key timestamp:(NSDate *) timestamp fault:(Fault
**)fault;

// async methods with responder
-(void)expireIn:(NSString *)key timeToLive:(int)seconds
responder:(id<IResponder>)responder;
-(void)expireAt:(NSString *)key timestamp:(NSDate *) timestamp
responder:(id<IResponder>)responder;

// async methods with block-based callback
-(void)expireIn:(NSString *)key timeToLive:(int)seconds response:(void
(^)(id))responseBlock error:(void (^)(Fault *))errorBlock;
-(void)expireAt:(NSString *)key timestamp:(NSDate *) timestamp response:(void
(^)(id))responseBlock error:(void (^)(Fault *))errorBlock;
```

where:

`key`	- identifies the object to extend the life of in cache.
`seconds`	- number of seconds to extend the life of object in cache by. Must be a value between 1 and 7200 (2 hours).
`timestamp`	- a date when the object should expire and removed from cache. The difference between `date` and the current time must be equal or less than 7200000 milliseconds (2 hours).
`responder, response, error`	- the callbacks used for asynchronous calls to deliver result or fault to the calling program.

Examples:

```
// sync

Fault *fault = nil;
BOOL result = [backendless.cache expireIn:@"CacheWeather" timeToLive:900
fault:&fault];
if (fault) {
```

```
    NSLog(@"FAULT: %@", fault);
    return;
}

NSLog(@"[SYNC] object life has been extended");

// async

NSDate *timestamp = [NSDate dateWithTimeIntervalSinceNow:3600];

[backendless.cache expireAt:@"CacheWeather" timestamp:timestamp
response:^(id result) {
       NSLog(@"[ASYNC] object life has been extended");

 }
 error:^(Fault *fault) {
       NSLog(@"FAULT: %@", fault);
 }];
```

Deleting object from cache

This method deleted an object from cache if it is present there. All methods are available via `backendless.cache` accessor:

```
// sync methods with fault option
-(BOOL)remove:(NSString *)key fault:(Fault **)fault;

// async methods with responder
-(void) remove:(NSString *)key responder:(id<IResponder>)responder;

// async methods with block-based callback
-(void) remove:(NSString *)key response:(void (^)(id))responseBlock error:(void
(^)(Fault *))errorBlock;
```

where:

`key`	- key assigned to the object to identify it in cache. The key is used to retrieve the object from cache or to check if the cache still contains the object.
`responder, response, error`	- the callbacks used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Examples:

```
// sync

Fault *fault = nil;
BOOL result = [backendless.cache remove:@"CacheWeather" fault:&fault];
if (fault) {
```

```
    NSLog(@"FAULT: %@", fault);
    return;
}

NSLog(@"[SYNC] object has been deleted from cache");


// async

[backendless.cache remove:@"CacheWeather"
response:^(id result) {
       NSLog(@"[ASYNC] object has been deleted from cache”);

 }
 error:^(Fault *fault) {
        NSLog(@"FAULT: %@", fault);
 }];
```