Atomic Counters API

Backendless Counters API provides a centralized server-side facility for working with values that may be updated atomically. Every counter has a name assigned to it. The name is used in all operations to identify the counter. Since the counter value is managed centrally, multiple heterogeneous clients can access and modify the value.

All counter APIs are available via `backendless.counters [methodname]` accessor. For example, the following code increments a counter and returns the current value:

```
Fault *fault = nil;
NSNumber *result = [backendless.counters incrementAndGet:@"MyCounter" fault:&fault];
```

Additionally, there is a shortcut approach with support for generics:

```
id <IAtomic> myCounter = [backendless.counters of:@"MyCounter"];
NSNumber *result = [myCounter incrementAndGet];
```

Increment by 1, return previous

Atomically increments by one the current value and returns the previous value of the counter. It is possible that multiple concurrent client requests may receive the same previous value. This occurs since only the incrementing part of the logic is atomic, the retrieval of the value before it is incremented is not.

```
// sync method with fault option
-(NSNumber *)getAndIncrement:(NSString *)counterName fault:(Fault **)fault;
// async method with responder
-(void)getAndIncrement:(NSString *)counterName responder:(id<IResponder>)responder;
// async methods with block-based callback
-(void)getAndIncrement:(NSString *)counterName response:(void (^)(NSNumber
*))responseBlock error:(void (^)(Fault *))errorBlock;
```

where:

`counterName`	- name of the counter to increment.
`responder, response, error`	- the callback used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Example:

```
// synchronous methods

Fault *fault = nil;
NSNumber *result = [backendless.counters getAndIncrement:@"MyCounter" fault:&fault];
```

```
if (fault) {
        NSLog(@"[FAULT] %@",  fault);
}

NSLog(@"[SYNC] previous counter value: %@",  result);

// asynchronous methods

[backendless.counters getAndIncrement:@"MyCounter"
response:^( NSNumber *result) {
        NSLog(@"[ASYNC] previous counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];

// IAtomic approach

id <IAtomic> counter = [backendless.counters of:@"MyCounter"];
[counter getAndIncrement:^( NSNumber *result) {
        NSLog(@"[ASYNC] previous counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];
```

Increment by 1, return current

Atomically increments by one the current value and returns the updated (current) value of the counter. Multiple concurrent client requests are guaranteed to return unique updated value.

```
// sync method with fault option
-(NSNumber *)incrementAndGet:(NSString *)counterName fault:(Fault **)fault;
// async method with responder
-(void) incrementAndGet:NSString *)counterName responder:(id<IResponder>)responder;
// async methods with block-based callback
-(void) incrementAndGet:NSString *)counterName response:(void (^)(NSNumber
*))responseBlock error:(void (^)(Fault *))errorBlock;
```

where:

`counterName`	- name of the counter to increment.
`responder, response, error`	- the callback used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Example:

```
// synchronous methods

Fault *fault = nil;
NSNumber *result = [backendless.counters incrementAndGet:@"MyCounter" fault:&fault];
if (fault) {
```

```
        NSLog(@"[FAULT] %@",  fault);
}

NSLog(@"[SYNC] current counter value: %@",  result);

// asynchronous methods

[backendless.counters incrementAndGet:@"MyCounter"
response:^( NSNumber *result) {
        NSLog(@"[ASYNC] current counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];

// IAtomic approach

id <IAtomic> counter = [backendless.counters of:@"MyCounter"];
[counter incrementAndGet:^( NSNumber *result) {
        NSLog(@"[ASYNC] current counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];
```

Decrement by 1, return previous

Atomically decrements by one the current value and returns the previous value of the counter. It is possible that multiple concurrent client requests may receive the same previous value. This occurs since only the decrementing part of the logic is atomic, the retrieval of the value before it is decremented is not.

```
// sync method with fault option
-(NSNumber *)getAndIDecrement:(NSString *)counterName fault:(Fault **)fault;
// async method with responder
-(void)getAndDecrement:(NSString *)counterName responder:(id<IResponder>)responder;
// async methods with block-based callback
-(void)getAndDecrement:(NSString *)counterName response:(void (^)(NSNumber
*))responseBlock error:(void (^)(Fault *))errorBlock;
```

where:

`counterName`	- name of the counter to decrement.
`responder, response, error`	- the callback used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Example:

```
// synchronous methods

Fault *fault = nil;
NSNumber *result = [backendless.counters getAndDecrement:@"MyCounter" fault:&fault];
```

```
if (fault) {
        NSLog(@"[FAULT] %@",  fault);
}

NSLog(@"[SYNC] previous counter value: %@",  result);

// asynchronous methods

[backendless.counters getAndDecrement:@"MyCounter"
response:^( NSNumber *result) {
        NSLog(@"[ASYNC] previous counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];

// IAtomic approach

id <IAtomic> counter = [backendless.counters of:@"MyCounter"];
[counter getAndDecrement:^( NSNumber *result) {
        NSLog(@"[ASYNC] previous counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];
```

Decrement by 1, return current

Atomically decrements by one the current value and returns the updated (current) value of the counter. Multiple concurrent client requests are guaranteed to return unique updated value.

```
// sync method with fault option
-(NSNumber *)decrementAndGet:(NSString *)counterName fault:(Fault **)fault;
// async method with responder
-(void) decrementAndGet:NSString *)counterName responder:(id<IResponder>)responder;
// async methods with block-based callback
-(void) decrementAndGet:NSString *)counterName response:(void (^)(NSNumber
*))responseBlock error:(void (^)(Fault *))errorBlock;
```

where:

`counterName`	- name of the counter to decrement.
`responder, response, error`	- the callback used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Example:

```
// synchronous methods

Fault *fault = nil;
NSNumber *result = [backendless.counters decrementAndGet:@"MyCounter" fault:&fault];
if (fault) {
```

```
        NSLog(@"[FAULT] %@",  fault);
}

NSLog(@"[SYNC] current counter value: %@",  result);

// asynchronous methods

[backendless.counters decrementAndGet:@"MyCounter"
response:^( NSNumber *result) {
        NSLog(@"[ASYNC] current counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];

// IAtomic approach

id <IAtomic> counter = [backendless.counters of:@"MyCounter"];
[counter decrementAndGet:^( NSNumber *result) {
        NSLog(@"[ASYNC] current counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];
```

Increment by N, return current

Atomically adds the given value to the current value and returns the updated (current) value of the counter. Multiple concurrent client requests are guaranteed to return updated value.

```
// sync method with fault option
-(NSNumber *)addAndGet:(NSString *)counterName value:(long)value fault:(Fault **)fault;
// async method with responder
-(void) addAndGet:NSString *)counterName value:(long)value
responder:(id<IResponder>)responder;
// async methods with block-based callback
-(void) addAndGet:NSString *)counterName value:(long)value response:(void
(^)(NSNumber *))responseBlock error:(void (^)(Fault *))errorBlock;
```

where:

`counterName`	- name of the counter to increment.
`value`	- number to add to the current counter value
`responder, response, error`	- the callback used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Example:

```
// synchronous methods
```

```
Fault *fault = nil;
NSNumber *result = [backendless.counters addAndGet:@"MyCounter" value:7 fault:&fault];
if (fault) {
        NSLog(@"[FAULT] %@",  fault);
}

NSLog(@"[SYNC] current counter value: %@",  result);

// asynchronous methods

[backendless.counters addAndGet:@"MyCounter" value:7
response:^( NSNumber *result) {
        NSLog(@"[ASYNC] current counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];

// IAtomic approach

id <IAtomic> counter = [backendless.counters of:@"MyCounter"];
[counter addAndGet:17
response:^( NSNumber *result) {
        NSLog(@"[ASYNC] current counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];
```

Increment by N, return previous

Atomically adds the given value to the current value and returns the previous value of the counter. It is possible that multiple concurrent client requests may receive the same previous value. This occurs since only the incrementing part of the logic is atomic, the retrieval of the value before it is incremented is not.

```
// sync method with fault option
-(NSNumber *)getAndAdd:(NSString *)counterName value:(long)value fault:(Fault **)fault;
// async method with responder
-(void) getAndAdd:NSString *)counterName value:(long)value
responder:(id<IResponder>)responder;
// async methods with block-based callback
-(void) getAndAdd:NSString *)counterName value:(long)value response:(void (^)(NSNumber
*))responseBlock error:(void (^)(Fault *))errorBlock;
```

where:

`counterName`	- name of the counter to increment.
`value`	- number to add to the current counter value
`responder, response, error`	- the callback used for asynchronous calls to indicate that the operation has either successfully completed or

resulted in error.

Example:

```
// synchronous methods

Fault *fault = nil;
NSNumber *result = [backendless.counters getAndAdd:@"MyCounter" value:7 fault:&fault];
if (fault) {
        NSLog(@"[FAULT] %@",  fault);
}

NSLog(@"[SYNC] previous counter value: %@",  result);

// asynchronous methods

[backendless.counters getAndAdd:@"MyCounter" value:7
response:^( NSNumber *result) {
        NSLog(@"[ASYNC] previous counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];

// IAtomic approach

id <IAtomic> counter = [backendless.counters of:@"MyCounter"];
[counter getAndAdd:17
response:^( NSNumber *result) {
        NSLog(@"[ASYNC] previous counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];
```

Conditional update

Atomically sets the value to the given updated value if the current value == the expected value.

```
// sync method with fault option
-(NSNumber *)compareAndSet:(NSString *)counterName expected:(long)expected
updated:(long)updated fault:(Fault **)fault;
// async method with responder
-(void) compareAndSet:NSString *)counterName counterName expected:(long)expected
updated:(long)updated responder:(id<IResponder>)responder;
// async methods with block-based callback
-(void) compareAndSet:NSString *)counterName counterName expected:(long)expected
updated:(long)updated response:(void (^)(NSNumber *))responseBlock error:(void (^)(Fault
*))errorBlock;
```

where:

`counterName` - name of the counter to compare.

`expected`	- the expected value of the counter. If the current value equals the expected value, the counter is set to the "`updated`" value.
`updated`	- the new value to assign to the counter if the current value equals the `expected` value.
`responder, response, error`	- the callback used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Example:

```
// synchronous methods

Fault *fault = nil;
NSNumber *result = [backendless.counters compareAndSet:@"MyCounter" expected:1000
updated:2000 fault:&fault];
if (fault) {
      NSLog(@"[FAULT] %@",  fault);
}

NSLog(@"[SYNC] value has been updated: %@",  result);

// asynchronous methods

[backendless.counters compareAndSet:@"MyCounter" expected:1000 updated:2000
response:^( NSNumber *result) {
      NSLog(@"[ASYNC] value has been updated: %@",  result);
 }
 error:^(Fault *fault) {
      NSLog(@"[FAULT] %@",  fault);
 }];

// IAtomic approach

id <IAtomic> counter = [backendless.counters of:@"MyCounter"];
[counter compareAndSet:1000 updated:2000
response:^( NSNumber *result) {
      NSLog(@"[ASYNC] value has been updated: %@",  result);
 }
 error:^(Fault *fault) {
      NSLog(@"[FAULT] %@",  fault);
 }];
```

Get current

Returns the current value of the counter.

```
// sync method with fault option
-(NSNumber *)get:(NSString *)counterName fault:(Fault **)fault;
```

```
// async method with responder
-(void) get:NSString *)counterName responder:(id<IResponder>)responder;
// async methods with block-based callback
-(void) get:NSString *)counterName response:(void (^)(NSNumber *))responseBlock
error:(void (^)(Fault *))errorBlock;
```

where:

`counterName`	- name of the counter to retrieve.
`responder, response, error`	- the callback used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Example:

```
// synchronous methods

Fault *fault = nil;
NSNumber *result = [backendless.counters get:@"MyCounter" fault:&fault];
if (fault) {
        NSLog(@"[FAULT] %@",  fault);
}

NSLog(@"[SYNC] current counter value: %@",  result);

// asynchronous methods

[backendless.counters get:@"MyCounter"
response:^( NSNumber *result) {
        NSLog(@"[ASYNC] current counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];

// IAtomic approach

id <IAtomic> counter = [backendless.counters of:@"MyCounter"];
[counter get:^( NSNumber *result) {
        NSLog(@"[ASYNC] current counter value: %@",  result);
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];
```

Reset

Resets the current counter value to zero.

```
// sync method with fault option
-(void) reset:(NSString *)counterName fault:(Fault **)fault;
// async method with responder
-(void) reset:NSString *)counterName responder:(id<IResponder>)responder;
// async methods with block-based callback
```

```
-(void) reset:NSString *)counterName response:(void (^)(NSNumber *))responseBlock
error:(void (^)(Fault *))errorBlock;
```

where:

`counterName`	- name of the counter to reset.
`responder, response, error`	- the callback used for asynchronous calls to indicate that the operation has either successfully completed or resulted in error.

Example:

```
// synchronous methods

Fault *fault = nil;
[backendless.counters reset:@"MyCounter" fault:&fault];
if (fault) {
        NSLog(@"[FAULT] %@",  fault);
}

NSLog(@"SYNC] counter has been reset");

// asynchronous methods

[backendless.counters get:@"MyCounter"
response:^( id result) {
        NSLog(@"[SYNC] counter has been reset");
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];

// IAtomic approach

id <IAtomic> counter = [backendless.counters of:@"MyCounter"];
[counter get:^(id result) {
        NSLog(@"SYNC] counter has been reset");
 }
 error:^(Fault *fault) {
        NSLog(@"[FAULT] %@",  fault);
 }];
```