Backendless UNITY SDK API Reference

2015-02-26

Acrodea,Inc.

Date	Version	Note
2015-02-26	1.0	初版

Backendless UNITY SDK API Document

概要

本ドキュメントは米国Backendless 社が提供するサービスをUnity プラットフォームから利用するためのSDKが提供するAPIについて説明したものです。

免責および再配布条件

- 本ドキュメント及び本ドキュメントが説明するソフトウェア・サービス・プログラムの使用は、これらの利用者（以下利用者と略す）自身の責任でなされるものとします。
- すべての資料、情報、ソフトウェア、プログラムおよびサービスは現状のまま提供され、いかなる保証も適用されません。株式会社アクロディアは、適用される法律の許す限りにおいて、法律上の瑕疵担保責任、商品性の保証、特定目的への適合性の保証、権利の不侵害の保証を含むいかなる明示もしくは黙示の保証責任も一切負いません。
- 利用者は、本ドキュメント及び本ドキュメントが説明するソフトウェア・サービス・プログラムを取得もしくは利用する場合においては、利用者ご自身の判断および責任において行っていただくこと、また、その結果として発生するデータの損失または利用者のコンピューター・システムへの損傷などのいかなる損害もすべて利用者の責任となることを理解し、同意するものとします。
- 本ドキュメントに記載された仕様は、予告無く変更される場合があります。
- 特に明記されている場合を除き、Backendlessサービスに関する全て（ソフトウェア、電子データ、画像、ドキュメント）の権利は米国Backendless社に帰属します。
- 本ドキュメントにて説明されているBackendless Unity SDK はApache 2.0 ライセンスに基づき株式会社アクロディアが作成したものであり、また本ドキュメントの著作権は株式会社アクロディアに帰属します。
- 本ドキュメントの複製、頒布、展示、実演は、 Creative Commons 表示- 継承 ライセンス（CC BYSA2.1 JP）に基づいて許可されます。http://creativecommons.org/licenses/by-sa/2.1/jp/
- 本ドキュメントが説明するソフトウェアのうち Backendless Unity SDK に関しては Apache License 2.0（http://www.apache.org/licenses/LICENSE-2.0.html） に基づいて再利用が可能です。

1 User Service

1.1. Retrieve User Entity Properties

- **Asynchronousの呼出関数:**

```
public void DescribeUserClass(AsyncCallback<List<UserProperty>> callback);
```

- **Synchronousの呼出関数:**

```
public List<UserProperty> DescribeUserClass();
```

- **Parameter:**

callback - Asynchronous 関数の結果を取得するcallback関数を指定します。

- **UserProperty classの定義:**

```
namespace BackendlessAPI.Property
{
  public class UserProperty
  {
    public bool IsIdentity { get; set; }
    public string Name { get; set; }
    public bool IsRequired { get; set; }
    public bool IsSelected { get; set; }
    public DateTypeEnum Type { get; set; }
    public object DefaultValue { get; set; }
  }
}
```

- **Asynchronousの呼出例:**

```
AsyncCallback<List<UserProperty>> callback = new AsyncCallback<List<UserProperty>>(
  props =>
  {
    foreach (UserProperty p in props)
    {
      Debug.Log("prop name " + p.Name);
      Debug.Log("\tis identity " + p.IsIdentity);
      Debug.Log("\tis required " + p.IsRequired);
      Debug.Log("\tprop type " + p.Type);
      Debug.Log("\tdefault value " + p.DefaultValue);
    }
  },
  fault =>
  {
    Debug.Log("Error - " + fault);
  });
Backendless.UserService.DescribeUserClass(callback);
```

♦ **Synchronousの呼出例:**

```
try
{
  List<UserProperty> props = Backendless.UserService.DescribeUserClass();
  foreach (UserProperty p in props)
  {
    Debug.Log("prop name " + p.Name);
    Debug.Log("\tis identity " + p.IsIdentity);
    Debug.Log("\tis required " + p.IsRequired);
    Debug.Log("\tprop type " + p.Type);
    Debug.Log("\tdefault value " + p.DefaultValue);
  }
}
catch (BackendlessException e)
{
  Debug.Log("Error - " + e);
}
```

1.2. User Registration

♦ **Asynchronousの呼出関数:**

```
public void Register(BackendlessUser user, AsyncCallback<BackendlessUser> callback);
```

♦ **Synchronousの呼出関数:**

```
public BackendlessUser Register(BackendlessUser user);
```

♦ **Parameter:**

user - アカウント登録に必要な属性値を含むBackendlessUser class variableを指定します。

callback - Asynchronous関数の結果を取得するcallback関数を指定します。

♦ **Asynchronousの呼出例:**

```
AsyncCallback<BackendlessUser> callback = new AsyncCallback<BackendlessUser>(
user =>
{
  Debug.Log("User registered. Assigned ID - " + user.UserId);
  Dictionary<string, object> props = user.Properties;

  foreach (KeyValuePair<string, object> pair in props)
    Debug.Log(String.Format("Property: {0} - {1}", pair.Key, pair.Value));
},
fault =>
{
  Debug.Log("Error - " + fault);
});

BackendlessUser newUser = new BackendlessUser();
newUser.Email = "jb@mi6.co.uk";
newUser.Password = "hard2guess";
newUser.SetProperty("login", "james.bond");
Backendless.UserService.Register(newUser, callback);
```

♦ **Synchronousの呼出例:**

```
try
{
  BackendlessUser newUser = new BackendlessUser();
  newUser.Email = "jb@mi6.co.uk";
  newUser.Password = "hard2guess";
  newUser.SetProperty("login", "james.bond");
  BackendlessUser user = Backendless.UserService.Register(newUser);
  Debug.Log("User registered. Assigned ID - " + user.UserId);
  Dictionary<string, object> props = user.Properties;

  foreach (KeyValuePair<string, object> pair in props)
    Debug.Log(String.Format("Property: {0} - {1}", pair.Key, pair.Value));
}
catch (BackendlessException e)
{
  Debug.Log("Error - " + e);
}
```

1.3. Login

♦ **Asynchronousの呼出関数:**

```
public void Login(string login, string password, AsyncCallback<BackendlessUser> callback);
```

♦ **Synchronousの呼出関数:**

```
public BackendlessUser Login(string login, string password);
```

♦ **Parameter:**

login - identityに指定した属性の値を指定します。

password – ユーザーのpasswordを指定します。

responder - Asynchronous関数の結果を取得するcallback関数を指定します。

♦ **Asynchronousの呼出例:**

```
AsyncCallback<BackendlessUser> callback = new AsyncCallback<BackendlessUser>(
  user =>
  {
    Debug.Log("User logged in. Assigned ID - " + user.UserId);
    Dictionary<string, object> props = user.Properties;
    foreach (KeyValuePair<string, object> pair in props)
      Debug.Log(String.Format("Property: {0} - {1}", pair.Key, pair.Value));
  },
  fault =>
  {
    Debug.Log("Error - " + fault);
  });

string login = "james.bond123";
string password = "guessIt";
Backendless.UserService.Login(login, password, callback);
```

♦ **Synchronousの呼出例:**

```
try
{
  string login = "james.bond123";
  string password = "guessIt";
  BackendlessUser user = Backendless.UserService.Login(login, password);
  Debug.Log("User logged in. Assigned ID - " + user.UserId);
  Dictionary<string, object> props = user.Properties;
  foreach (KeyValuePair<string, object> pair in props)
    Debug.Log(String.Format("Property: {0} - {1}", pair.Key, pair.Value));
}
catch (BackendlessException e)
{
  Debug.Log("Error - " + e);
}
```

1.4. Update User Properties

♦ **Asynchronousの呼出関数:**

```
public void Update(BackendlessUser user, AsyncCallback<BackendlessUser> callback);
```

♦ **Synchronousの呼出関数:**

```
public BackendlessUser Update(BackendlessUser user);
```

♦ **Parameter:**

user - アカウント更新に必要な属性値を含むBackendlessUser class variableを指定します。

responder - Asynchronous関数の結果を取得するcallback関数を指定します。

♦ **Asynchronousの呼出例:**

```
AsyncCallback<BackendlessUser> updateCallback = new AsyncCallback<BackendlessUser>(
  user =>
  {
    Debug.Log("User account has been updated");
    Dictionary<string, object> props = user.Properties;
    foreach (KeyValuePair<string, object> pair in props)
      Debug.Log(String.Format("Property: {0} - {1}", pair.Key, pair.Value));
  },
  fault =>
  {
    Debug.Log(fault.ToString());
  });

AsyncCallback<BackendlessUser> loginCallback = new AsyncCallback<BackendlessUser>(
  user =>
  {
    user.SetProperty("phoneNumber", "5551212");
    Backendless.UserService.Update(user, updateCallback);
  },
  fault =>
```

```
  {
    Debug.Log(fault.ToString());
  });

Backendless.UserService.Login(login, password, loginCallback);
```

- **Synchronousの呼出例:**

```
BackendlessUser user;

try
{
  user = Backendless.UserService.Login(login, password);
}
catch (BackendlessException exception)
{
  // login failed, to get the error code, call exception.Fault.FaultCode
}

try
{
  user.SetProperty("phoneNumber", "5551212");
  user = Backendless.UserService.Update(user);
}
catch (BackendlessException exception)
{
  // update failed, to get the error code, call exception.Fault.FaultCode
}
```

1.5. Get Current User

Login中のユーザーのBackendlessUser class pointerを取得します。

```
public BackendlessUser Backendless.UserService.CurrentUser
{
  get;
}
```

Login状態でない場合にはNULLを返します。

1.6. Logout

- **Asynchronousの呼出関数:**

```
public void Logout(AsyncCallback<object> callback);
```

- **Synchronousの呼出関数:**

```
public void Logout();
```

- **Parameter:**

callback - Asynchronous 関数の結果を取得するcallback関数を指定します。

♦ **Asynchronousの呼出例:**

```
AsyncCallback<object> logoutCallback = new AsyncCallback<object>(
  user =>
  {
    Debug.Log("User has been logged out");
  },
  fault =>
  {
    Debug.Log(fault.ToString());
  });

AsyncCallback<BackendlessUser> loginCallback = new AsyncCallback<BackendlessUser>(
  user =>
  {
    Backendless.UserService.Logout(logoutCallback);
  },
  fault =>
  {
    Debug.Log(fault.ToString());
  });

Backendless.UserService.Login(login, password, loginCallback);
```

♦ **Synchronousの呼出例:**

```
BackendlessUser user;

try
{
  user = Backendless.UserService.Login(login, password);
}
catch (BackendlessException exception)
{
  // login failed, to get the error code, call exception.Fault.FaultCode
}

try
{
  // now log out:
  Backendless.UserService.Logout();
}
catch (BackendlessException exception)
{
  // logout failed, to get the error code, call exception.Fault.FaultCode
}
```

1.7. Password Recovery

♦ **Asynchronousの呼出関数:**

```
public void RestorePassword(string identity, AsyncCallback<object> callback);
```

♦ **Synchronousの呼出関数:**

```
public void RestorePassword(string identity);
```

♦ **Parameter:**

identity - identityに指定した属性の値を指定します。

responder - Asynchronous関数の結果を取得するcallback関数を指定します。

- **Asynchronousの呼出例:**

```
AsyncCallback<object> recoveryCallback = new AsyncCallback<object>(
  user =>
  {
    Debug.Log("Password recovery email has been sent");
  },
  fault =>
  {
    Debug.Log(fault.ToString());
  });

Backendless.UserService.RestorePassword(identity, recoveryCallback);
```

- **Synchronousの呼出例:**

```
try
{
  Backendless.UserService.RestorePassword(identity);
  Debug.Log("Password recovery email has been sent");
}
catch (BackendlessException exception)
{
  // password recovery failed, to get the error code, call exception.Fault.FaultCode
}
```

1.8. Role To User Mapping

AssignRoleとUnassignRoleはセキュリティ的な観点からクライアントサイドのAPIから削除されました。

2 Data Service

2.1. Data Object

Backendless unity3d SDKのData Objectはc# class objectをサポートします。.

すべてのクラス objectに共通で必要となる属性をBackendlessEntity クラスとして定義しました。
Data Objectを新規に定義する場合はこのBackendlessEntity クラスを承継し、必要な属性を追加して使用してください。

```
class Address : BackendlessEntity
{
  public string Street { get; set; }
  public string City { get; set; }
  public string State { get; set; }
}

class Contact : BackendlessEntity
{
  public string Name { get; set; }
  public int Age { get; set; }
  public string Phone { get; set; }
  public string Title { get; set; }
  public bool IsValidate { get; set; }
  public long Expire { get; set; }
  public double Point { get; set; }
  public DateTime Birthday { get; set; }
  public Address Address { get; set; }
}

class PhoneBook : BackendlessEntity
{
  public Contact Owner { get; set; }
  public List<Contact> Contacts { get; set; }
}

※ JsonProperty attributeを使用してcustom column nameを指定できます。
   [JsonProperty("point")]
  public double Point { get; set; }

※ null値を持つpropertyは Nullable typeで指定できます。
   public double? Point { get; set; }
  public DateTime? Birthday { get; set; }
```

2.2. Saving Data Objects

- **Asynchronousの呼出関数:**

```
public void Backendless.Persistence.Of<T>().Save(T entity, AsyncCallback<T> callback);
```

- **Synchronousの呼出関数:**

```
public T Backendless.Persistence.Of<T>().Save(T entity);
```

♦ **Parameter:**

entity - 格納するclass objectを指定します。

callback - Asynchronous 関数の結果を取得するcallback関数を指定します。

♦ **Asynchronousの呼出例:**

```
class Contact : BackendlessEntity
{
  public string Name { get; set; }
  public int Age { get; set; }
  public string Phone { get; set; }
  public string Title { get; set; }
}

Contact contact = new Contact { Name = "Jack Daniels", Age = 147, Phone = "777-777-777", Title
    = "Favorites" };

AsyncCallback<Contact> callback = new AsyncCallback<Contact>(
  savedContact =>
  {
    Debug.Log("Saved = " + savedContact.Name);
  },
  fault =>
  {
    Debug.Log(fault);
  });

Backendless.Persistence.Of<Contact>().Save(contact, callback);
```

♦ **Synchronousの呼出例:**

```
try
{
  Contact savedContact = Backendless.Persistence.Of<Contact>().Save(contact);
  Debug.Log("Saved = " + savedContact.Name);
}
catch(BackendlessException e)
{
  Debug.Log(e);
}
```

2.3. Updating Data Objects

♦ **Asynchronousの呼出関数:**

```
public void Backendless.Persistence.Of<T>().Save(T entity, AsyncCallback<T> callback);
```

♦ **Synchronousの呼出関数:**

```
public T Backendless.Persistence.Of<T>().Save(T entity);
```

♦ **Parameter:**

entity - アップデートするclass objectを指定します。

callback - Asynchronous関数の結果を取得するcallback関数を指定します。

♦ **Asynchronousの呼出例:**

```
Contact contact = new Contact { Name = "Jack Daniels", Age = 147, Phone = "777-777-777", Title
    = "Favorites" };

AsyncCallback<Contact> callback = new AsyncCallback<Contact>(
  savedContact =>
  {
    Debug.Log("savedContact Title=" + savedContact.Title + " Phone=" + savedContact.Phone);

    savedContact.Title = "Most favorite";
    savedContact.Phone = "666-666-666";

    AsyncCallback<Contact> updateCallback = new AsyncCallback<Contact>(
      updateContact =>
      {
        Debug.Log("updateContact Title=" + updateContact.Title + " Phone=" +
    updateContact.Phone);
      },
      fault =>
      {
        Debug.Log(fault);
      });

    Backendless.Persistence.Of<Contact>().Save(savedContact, updateCallback);
  },
  fault =>
  {
    Debug.Log(fault);
  });

Backendless.Persistence.Of<Contact>().Save(contact, callback);
```

♦ **Synchronousの呼出例:**

```
try
{
  Contact savedContact = Backendless.Persistence.Of<Contact>().Save(contact);
  Debug.Log("savedContact Title=" + savedContact.Title + " Phone=" + savedContact.Phone);
  savedContact.Title = "Most favorite";
  savedContact.Phone = "666-666-666";
  Contact updateContact = Backendless.Persistence.Of<Contact>().Save(savedContact);
  Debug.Log("updateContact Title=" + updateContact.Title + " Phone=" + updateContact.Phone);
}
catch(BackendlessException e)
{
  Debug.Log(e);
}
```

2.4. Deleting Data Objects

♦ **Asynchronousの呼出関数:**

```
public void Backendless.Persistence.Of<T>().Remove(T entity, AsyncCallback<long> callback);
```

♦ **Synchronousの呼出関数:**

```
public long Backendless.Persistence.Of<T>().Remove(T entity);
```

♦ **Parameter:**

entity - 削除するclass objectを指定します。

callback - Asynchronous関数の結果を取得するcallback関数を指定します。

♦ **Asynchronousの呼出例:**

```
AsyncCallback<Contact> callback = new AsyncCallback<Contact>(
  savedContact =>
  {
    AsyncCallback<long> removeCallback = new AsyncCallback<long>(
      deletionTime =>
      {
        Debug.Log("deletionTime=" + deletionTime);
      },
      fault =>
      {
        Debug.Log(fault);
      });

    Backendless.Persistence.Of<Contact>().Remove(savedContact, removeCallback);
  },
  fault =>
  {
    Debug.Log(fault);
  });

Backendless.Persistence.Of<Contact>().Save(contact, callback);
```

♦ **Synchronousの呼出例:**

```
try
{
  Contact savedContact = Backendless.Persistence.Of<Contact>().Save(contact);
  long deletionTime = Backendless.Persistence.Of<Contact>().Remove(savedContact);
  Debug.Log("deletionTime=" + deletionTime);
}
catch(BackendlessException e)
{
  Debug.Log(e);
}
```

2.5. Basic Search

Basic Searchはクエリーを用いずに、先頭や末尾指定、ID指定等でオブジェクトを取得できます。

♦ **Asynchronousの呼出関数:**

```
public void Backendless.Persistence.Of<T>().Find(AsyncCallback<BackendlessCollection<T>>
    callback);
```

```
public void Backendless.Persistence.Of<T>().FindFirst(AsyncCallback<T> callback);
public void Backendless.Persistence.Of<T>().FindFirst(IList<string> relations,
    AsyncCallback<T> callback);
public void Backendless.Persistence.Of<T>().FindLast(AsyncCallback<T> callback);
public void Backendless.Persistence.Of<T>().FindLast(IList<string> relations, AsyncCallback<T>
    callback);
public void Backendless.Persistence.Of<T>().FindById(string id, AsyncCallback<T> callback);
public void Backendless.Persistence.Of<T>().FindById(string id, IList<string> relations, Async
    Callback<T> callback);
```

- **Synchronousの呼出関数:**

```
public BackendlessCollection<T> Backendless.Persistence.Of<T>().Find();
public T Backendless.Persistence.Of<T>().FindFirst();
public T Backendless.Persistence.Of<T>().FindFirst(IList<string> relations);
public T Backendless.Persistence.Of<T>().FindLast();
public T Backendless.Persistence.Of<T>().FindLast(IList<string> relations);
public T Backendless.Persistence.Of<T>().FindById(string id);
public T Backendless.Persistence.Of<T>().FindById(string id, IList<string> relations);
```

- **Parameter:**

relations – 検索対象となるrelation項目を指定します。

id - 検索するdataのobjectidを指定します。

callback - Asynchronous関数の結果を取得するcallback関数を指定します。

- **検索例：**

- 全てのContactの取得例

Asynchronousの呼出例:

```
AsyncCallback<BackendlessCollection<Contact>> callback = new
    AsyncCallback<BackendlessCollection<Contact>>(
  findContacts =>
  {
    foreach (Contact c in findContacts.GetCurrentPage())
    {
      Debug.Log("Find contact : " + c.Name);
    }
  },
  fault =>
  {
    Debug.Log(fault);
  });

Backendless.Persistence.Of<Contact>().Find(callback);
```

Synchronousの呼出例:

```
try
{
  BackendlessCollection<Contact> findContacts = Backendless.Persistence.Of<Contact>().Find();
  foreach (Contact c in findContacts.GetCurrentPage())
  {
```

```
    Debug.Log("Find contact : " + c.Name);
  }
}
catch(BackendlessException e)
{
  Debug.Log(e);
}
```

- Find First での取得例

Asynchronousの呼出例:

```
AsyncCallback<Contact> callback = new AsyncCallback<Contact>(
  firstContact =>
  {
    Debug.Log("First contact : " + firstContact.Name);
  },
  fault =>
  {
    Debug.Log(fault);
  });

Backendless.Persistence.Of<Contact>().FindFirst(callback);
```

Synchronousの呼出例:

```
try
{
  Contact firstContact = Backendless.Persistence.Of<Contact>().FindFirst();
  Debug.Log("First contact : " + firstContact.Name);
}
catch(BackendlessException e)
{
  Debug.Log(e);
}
```

- Find Last での取得例

Asynchronousの呼出例:

```
AsyncCallback<Contact> callback = new AsyncCallback<Contact>(
  lastContact =>
  {
    Debug.Log("Last contact : " + lastContact.Name);
  },
  fault =>
  {
    Debug.Log(fault);
  });

Backendless.Persistence.Of<Contact>().FindLast(callback);
```

Synchronousの呼出例:

```
try
{
  Contact lastContact = Backendless.Persistence.Of<Contact>().FindLast();
```

```
  Debug.Log("Last contact : " + lastContact.Name);
}
catch(BackendlessException e)
{
  Debug.Log(e);
}
```

- Find by IDでの取得例

Asynchronousの呼出例:

```
Contact contact = new Contact { Name = "Jack Daniels", Age = 147, Phone = "777-777-777", Title
    = "Favorites" };

AsyncCallback<Contact> callback = new AsyncCallback<Contact>(
  savedContact =>
  {
    AsyncCallback<Contact> findCallback = new AsyncCallback<Contact>(
      findContact =>
      {
        Debug.Log("Find contact : " + findContact.Name);
      },
      fault =>
      {
        Debug.Log(fault);
      });

    Backendless.Persistence.Of<Contact>().FindById(savedContact.ObjectId, findCallback);
  },
  fault =>
  {
    Debug.Log(fault);
  });

Backendless.Persistence.Of<Contact>().Save(contact, callback);
```

Synchronous の呼出例:

```
try
{
  Contact savedContact = Backendless.Persistence.Of<Contact>().Save(contact);
  Contact findContact = Backendless.Persistence.Of<Contact>().FindById(savedContact.ObjectId);
  Debug.Log("Find contact : " + findContact.Name);
}
catch(BackendlessException e)
{
  Debug.Log(e);
}
```

2.6. Advanced Search

BackendlessDataQueryオブジェクトを利用することで複雑な条件の検索が可能です。

- **Asynchronousの呼出関数:**

```
public void Backendless.Persistence.Of<T>().Find(BackendlessDataQuery dataQuery,
    AsyncCallback<BackendlessCollection<T>> callback);
```

♦ **Synchronousの呼出関数:**

```
public BackendlessCollection<T> Backendless.Persistence.Of<T>().Find(BackendlessDataQuery
    dataQuery);
```

♦ **Parameter:**

dataQuery – 検索queryを設定したBackendlessDataQuery instanceを指定します。

callback - Asynchronous関数の結果を取得するcallback関数を指定します。

♦ **BackendlessDataQuery classの定義:**

```
namespace BackendlessAPI.Persistence
{
  public class BackendlessDataQuery
  {
    public BackendlessDataQuery();
    public BackendlessDataQuery(List<string> properties);
    public BackendlessDataQuery(string whereClause);
    public BackendlessDataQuery(QueryOptions queryOptions);
    public BackendlessDataQuery(List<string> properties, string whereClause, QueryOptions
    queryOptions);

    public string WhereClause { get; set; }
    public QueryOptions QueryOptions { get; set; }
    public List<string> Properties { get; set; }
    public int PageSize { get; set; }
    public int Offset { get; set; }
  }
}
```

♦ **QueryOptions classの定義:**

```
namespace BackendlessAPI.Persistence
{
  public class QueryOptions
  {
    public QueryOptions();
    public QueryOptions(int pageSize, int offset);
    public QueryOptions(int pageSize, int offset, string sortBy);

    public int PageSize { get; set; }
    public int Offset { get; set; }
    public List<String> SortBy { get; set; }
    public List<string> Related { get; set; }
  }
}
```

♦ **検索例:**

```
Contact contact = new Contact { Name = "Jack Daniels", Age = 147, Phone = "777-777-777", Title
    = "Favorites" };

AsyncCallback<Contact> callback = new AsyncCallback<Contact>(
  savedContact =>
  {
    Debug.Log("Saved = " + savedContact.Name);
  },
```

```
  fault =>
  {
    Debug.Log(fault);
  });

Backendless.Persistence.Of<Contact>().Save(contact, callback);
```

- “age”属性が47である contact を全て取得

```
BackendlessDataQuery query = new BackendlessDataQuery();
query.WhereClause = "Age = 147";

AsyncCallback<BackendlessCollection<Contact>> callback = new
    AsyncCallback<BackendlessCollection<Contact>>(
  findContacts =>
  {
    foreach (Contact c in findContacts.GetCurrentPage())
    {
      Debug.Log("Find contact : " + c.Name);
    }
  },
  fault =>
  {
    Debug.Log(fault);
  });

Backendless.Persistence.Of<Contact>().Find(query, callback);
```

- “age” 属性が 21 より大きい contact を全て取得

```
BackendlessDataQuery query = new BackendlessDataQuery();
query.WhereClause = "Age > 21";

AsyncCallback<BackendlessCollection<Contact>> callback = new
    AsyncCallback<BackendlessCollection<Contact>>(
  findContacts =>
  {
    foreach (Contact c in findContacts.GetCurrentPage())
    {
      Debug.Log("Find contact : " + c.Name);
    }
  },
  fault =>
  {
    Debug.Log(fault);
});

Backendless.Persistence.Of<Contact>().Find(query, callback);
```

- “age” 属性が 21 以上 31 以下のものを全て取得。

```
BackendlessDataQuery query = new BackendlessDataQuery();
query.WhereClause = "Age >=21 AND Age <=30";

AsyncCallback<BackendlessCollection<Contact>> callback = new
    AsyncCallback<BackendlessCollection<Contact>>(
  findContacts =>
  {
    foreach (Contact c in findContacts.GetCurrentPage())
    {
      Debug.Log("Find contact : " + c.Name);
    }
  },
  fault =>
```

```
  {
    Debug.Log(fault);
});

Backendless.Persistence.Of<Contact>().Find(query, callback);
```

- ”Name”で取得

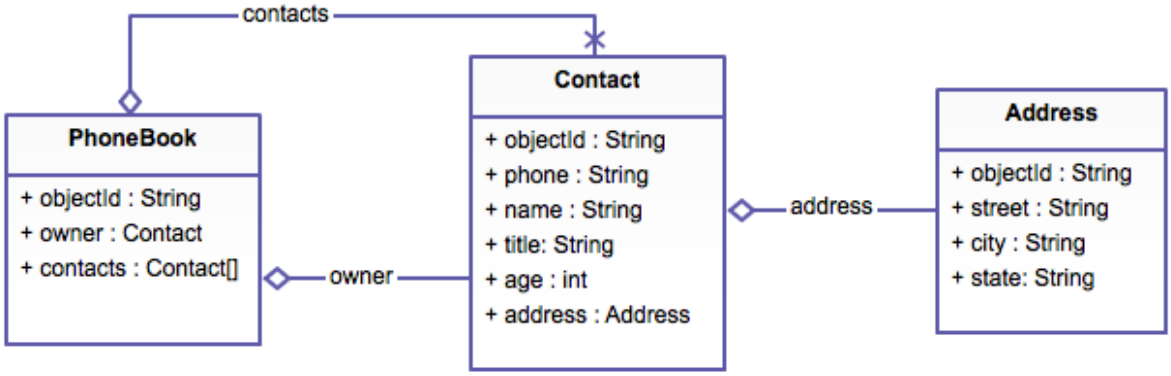
```
BackendlessDataQuery query = new BackendlessDataQuery();
query.WhereClause = "Name = 'Jack Daniels'";

AsyncCallback<BackendlessCollection<Contact>> callback = new
    AsyncCallback<BackendlessCollection<Contact>>(
  findContacts =>
  {
    foreach (Contact c in findContacts.GetCurrentPage())
    {
      Debug.Log("Find contact : " + c.Name);
    }
  },
  fault =>
  {
    Debug.Log(fault);
});

Backendless.Persistence.Of<Contact>().Find(query, callback);
```

- “Name” の部分一致で取得

```
BackendlessDataQuery query = new BackendlessDataQuery();
query.WhereClause = "Name LIKE 'Jack%'";

AsyncCallback<BackendlessCollection<Contact>> callback = new
    AsyncCallback<BackendlessCollection<Contact>>(
  findContacts =>
  {
    foreach (Contact c in findContacts.GetCurrentPage())
    {
      Debug.Log("Find contact : " + c.Name);
    }
  },
  fault =>
  {
    Debug.Log(fault);
});

Backendless.Persistence.Of<Contact>().Find(query, callback);
```

2.7. テーブル間でのリレーショナル操作

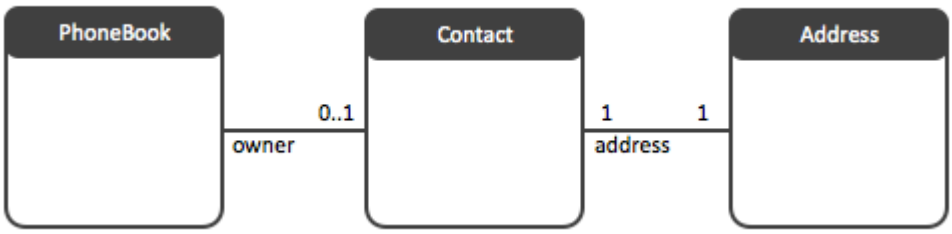
本節では以下のようなデータを想定して各種のオペレーション例をご説明します。




```
class Address : BackendlessEntity
{
  public string Street { get; set; }
  public string City { get; set; }
  public string State { get; set; }
}

class Contact : BackendlessEntity
{
  public string Name { get; set; }
  public int Age { get; set; }
  public string Phone { get; set; }
  public string Title { get; set; }
  public Address Address { get; set; }
}

class PhoneBook : BackendlessEntity
{
  public Contact Owner { get; set; }
  public List<Contact> Contacts { get; set; }
}
```

♦ **セーブとアップデート:**

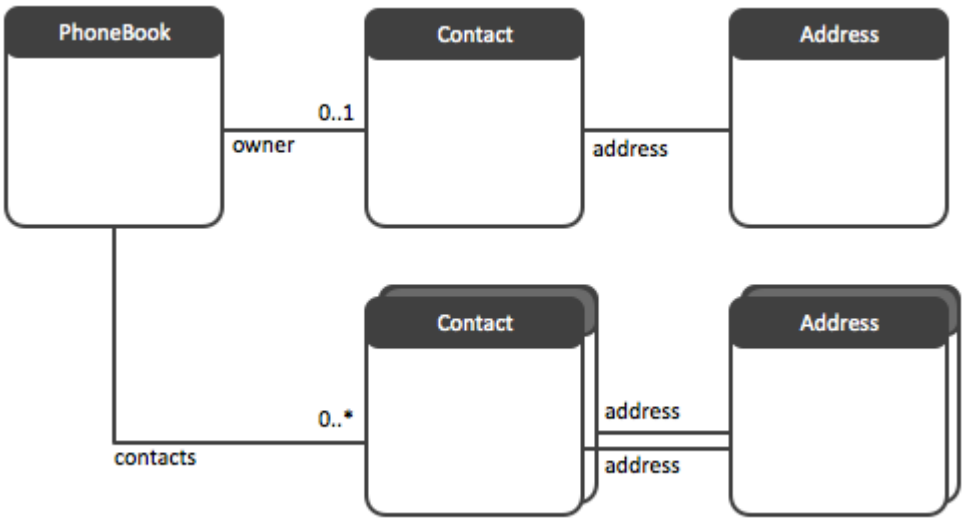
- Contactを持たないPhoneBookの保存例。




```
Address address = new Address();
address.Street = "TN 55";
address.City = "Lynchburg";
address.State = "Tennessee";

Contact owner = new Contact();
owner.Name = "Jack Daniels";
owner.Age = 147;
owner.Phone = "777-777-777";
owner.Title = "Favorites";
owner.Address = address;
```

```
PhoneBook phoneBook = new PhoneBook();
phoneBook.Owner = owner;

AsyncCallback<PhoneBook> callback = new AsyncCallback<PhoneBook>(
  savedPhoneBook =>
  {
  },
  fault =>
  {
  });

Backendless.Persistence.Of<PhoneBook>().Save(phoneBook, callback);
```

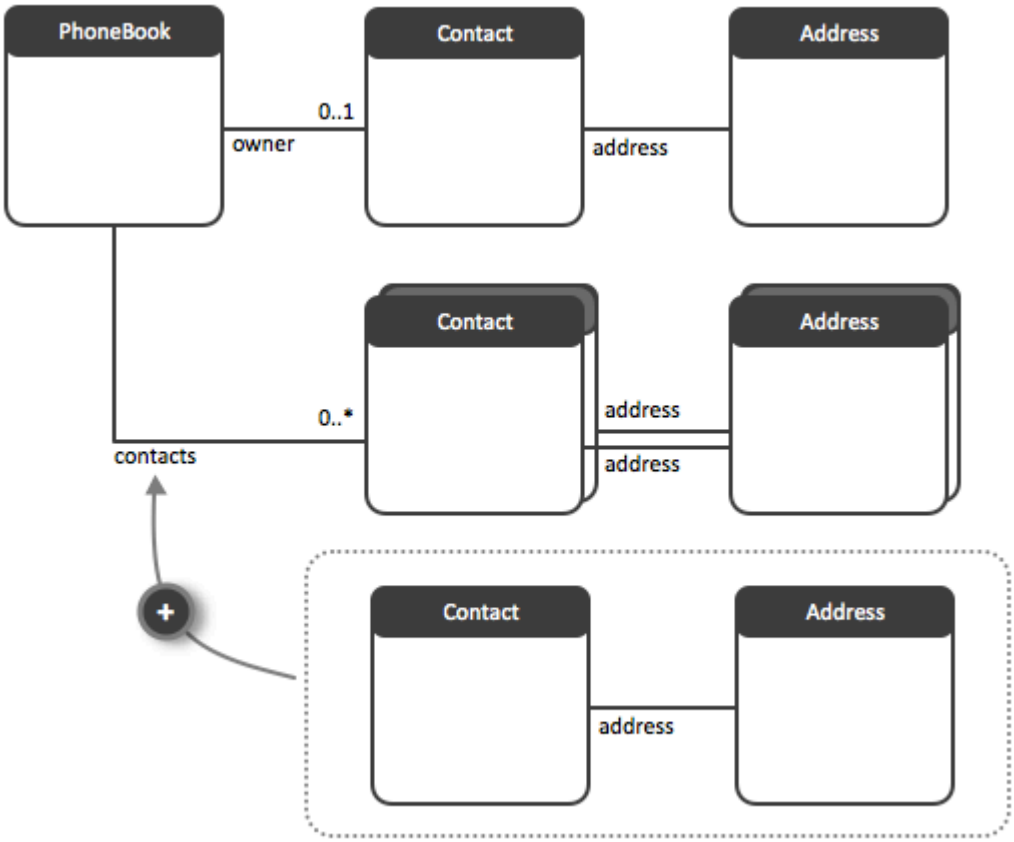
- ２つのContactを持つPhoneBookの保存例。




```
Address address = new Address();
address.Street = "TN 55";
address.City = "Lynchburg";
address.State = "Tennessee";

Contact owner = new Contact();
owner.Name = "Jack Daniels";
owner.Age = 147;
owner.Phone = "777-777-777";
owner.Title = "Favorites";
owner.Address = address;

Address contact1Address = new Address();
contact1Address.Street = "Main St";
contact1Address.City = "Smallville";
contact1Address.State = "Kansas";

Contact contact1 = new Contact();
contact1.Name = "Clark Kent";
contact1.Age = 75;
contact1.Phone = "111-111-111";
contact1.Title = "Super heroes";
contact1.Address = contact1Address;

Address contact2Address = new Address();
contact2Address.Street = "Cavern-X";
contact2Address.City = "Sedona";
contact2Address.State = "Arizona";

Contact contact2 = new Contact();
```

```
contact2.Name = "Wade Winston Wilson";
contact2.Age = 22;
contact2.Phone = "222-222-222";
contact2.Title = "Super heroes";
contact2.Address = contact2Address;

List<Contact> contacts = new List<Contact>();
contacts.Add(contact1);
contacts.Add(contact2);

PhoneBook phoneBook = new PhoneBook();
phoneBook.Owner = owner;
phoneBook.Contacts = contacts;

AsyncCallback<PhoneBook> callback = new AsyncCallback<PhoneBook>(
  savedPhoneBook =>
  {
  },
  fault =>
  {
  });

Backendless.Persistence.Of<PhoneBook>().Save(phoneBook, callback);
```

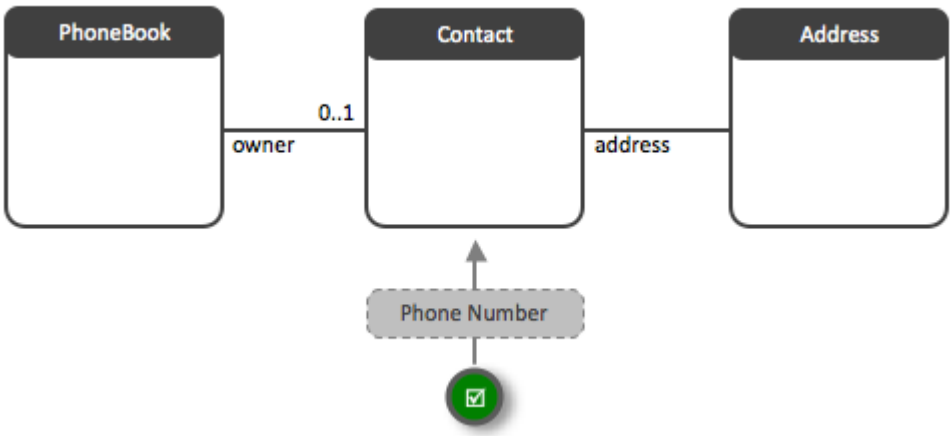
- 既に存在するPhoneBookにContactを追加（上の例に追加しています）




```
// skipped code for brevity..
// use the code from the example above
```

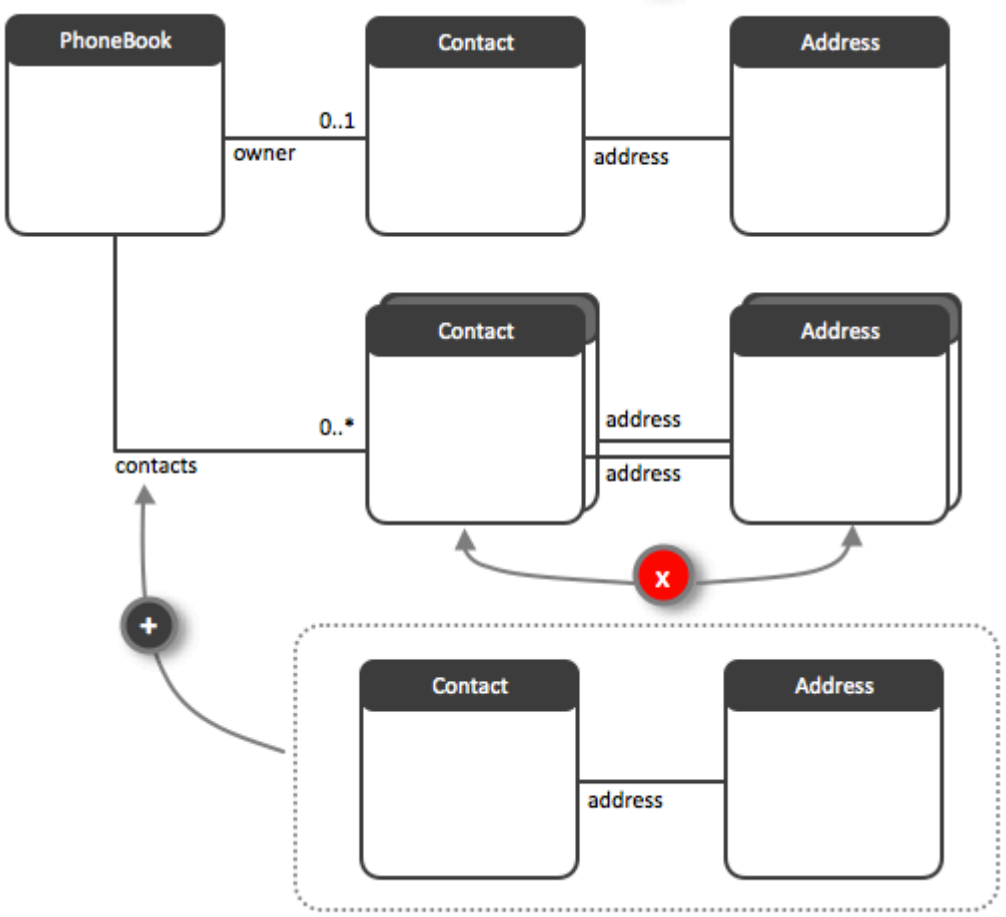
```
AsyncCallback<PhoneBook> callback = new AsyncCallback<PhoneBook>(
  savedPhoneBook =>
  {
    savedPhoneBook.Contacts.Add(contact2);

    AsyncCallback<PhoneBook> updateCallback = new AsyncCallback<PhoneBook>(
      updatedPhoneBook =>
      {
      },
      fault =>
      {
      });

    Backendless.Persistence.Of<PhoneBook>().Save(savedPhoneBook, updateCallback);
  },
  fault =>
  {
  });

Backendless.Persistence.Of<PhoneBook>().Save(phoneBook, callback);
```

- PhoneBook内部の属性（Owner）が保持するContactクラスの電話番号を更新。



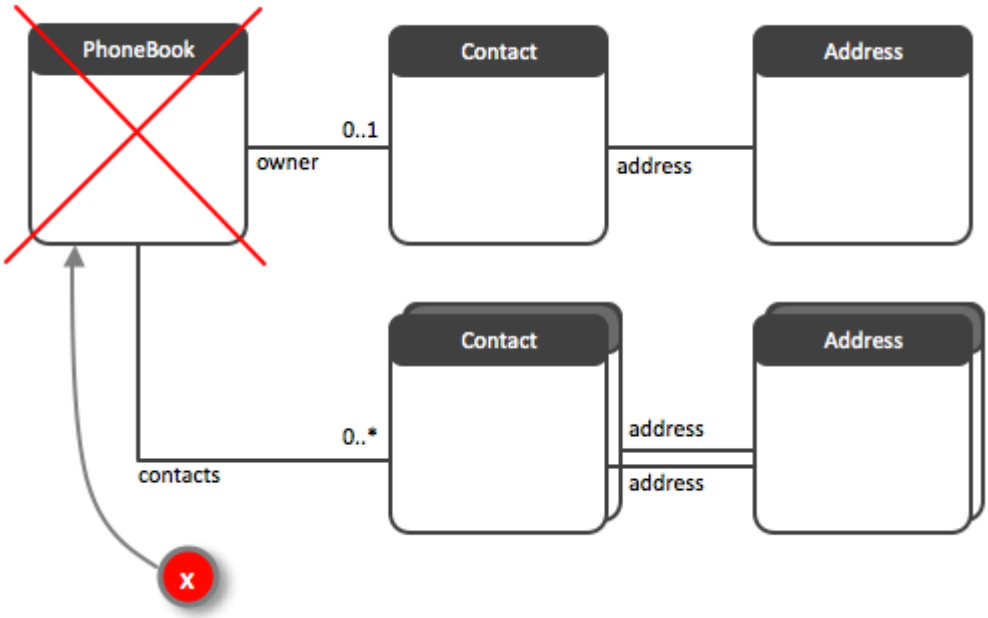

```
// use any of the samples above to the point when PhoneBook is saved,
// then use the savedPhoneBook instance:

savedPhoneBook.Owner.Phone = "555-555-555";

AsyncCallback<PhoneBook> updateCallback = new AsyncCallback<PhoneBook>(
  updatedPhoneBook =>
  {
  },
  fault =>
  {
  });

Backendless.Persistence.Of<PhoneBook>().Save(savedPhoneBook, updateCallback);
```

- PhoneBookからContactを一つ削除し、別のものを追加後に保存



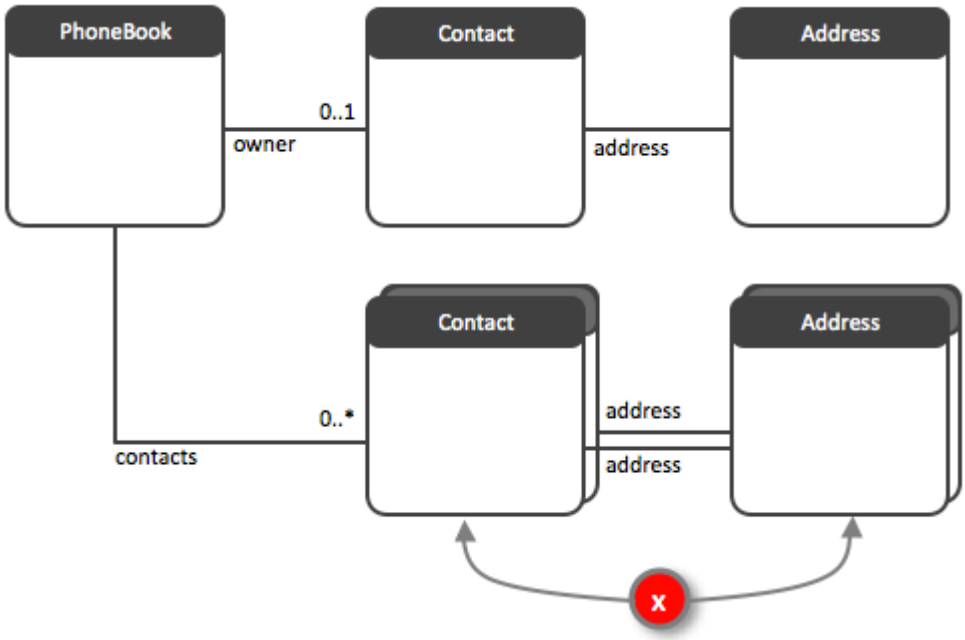

```
// use any of the samples above to the point when PhoneBook is saved,
// then use the savedPhoneBook instance:

Address contact2Address = new Address();
contact2Address.Street = "Cavern-X";
contact2Address.City = "Sedona";
contact2Address.State = "Arizona";

Contact contact2 = new Contact();
contact2.Name = "Wade Winston Wilson";
contact2.Age = 22;
contact2.Phone = "222-222-222";
contact2.Title = "Super heroes";
contact2.Address = contact2Address;

savedPhoneBook.Contacts.RemoveAt(0);
savedPhoneBook.Contacts.Add(contact2);
AsyncCallback<PhoneBook> updateCallback = new AsyncCallback<PhoneBook>(
  updatedPhoneBook =>
  {
  },
  fault =>
  {
  });

Backendless.Persistence.Of<PhoneBook>().Save(savedPhoneBook, updateCallback);
```

- 親オブジェクトを削除する例(PhoneBook の削除):




```
// use any of the samples above to the point when PhoneBook is saved,
// then use the savedPhoneBook instance:

AsyncCallback<long> removeCallback = new AsyncCallback<long>(
  deletionTime =>
  {
  },
  fault =>
  {
  });

Backendless.Persistence.Of<PhoneBook>().Remove(savedPhoneBook, removeCallback);
```

- Phonebook からContactを削除して結果を反映:




```
// use any of the samples above. Take the "savedPhoneBook" object
// and remove a contact from it:

savedPhoneBook.Contacts.RemoveAt(0);
AsyncCallback<PhoneBook> updateCallback = new AsyncCallback<PhoneBook>(
  updatedPhoneBook =>
  {
  },
  fault =>
  {
  });

Backendless.Persistence.Of<PhoneBook>().Save(savedPhoneBook, updateCallback);
```

♦ **サーチ**

- 特定のPhoneBook で “SmallVille” をAddress.Cityにもつ contacts を取得。

```
// use a sample above where PhoneBook is created with multiple contacts,
// then use the savedPhoneBook instance:

BackendlessDataQuery query = new BackendlessDataQuery();
query.WhereClause = "PhoneBook[Contacts].objectId='" + savedPhoneBook.ObjectId;
query.WhereClause += "' and ";
query.WhereClause += "Address.City = 'Smallville'";

AsyncCallback<BackendlessCollection<Contact>> findCallback = new
    AsyncCallback<BackendlessCollection<Contact>>(
  collection =>
  {
  },
  fault =>
  {
  });
Backendless.Persistence.Of<Contact>().Find(query, findCallback);
```

- 前述の例でCityの名称に “a” を含んでいるとする場合。

```
// use a sample above where PhoneBook is created with multiple contacts,
// then use the savedPhoneBook instance:

BackendlessDataQuery query = new BackendlessDataQuery();
query.WhereClause = "PhoneBook[Contacts].objectId='" + savedPhoneBook.ObjectId;
query.WhereClause += "' and ";
query.WhereClause += "Address.City like '%a%'";

AsyncCallback<BackendlessCollection<Contact>> findCallback = new
    AsyncCallback<BackendlessCollection<Contact>>(
  collection =>
  {
  },
  fault =>
  {
  });
Backendless.Persistence.Of<Contact>().Find(query, findCallback);
```

- 特定 phonebook 内で age が２０ より大きいものを取得。

```
// use a sample above where PhoneBook is created with multiple contacts,
// then use the savedPhoneBook instance:

BackendlessDataQuery query = new BackendlessDataQuery();
query.WhereClause = "PhoneBook[Contacts].objectId='" + savedPhoneBook.ObjectId;
query.WhereClause += "' and ";
query.WhereClause += "Age > 20";

AsyncCallback<BackendlessCollection<Contact>> findCallback = new
    AsyncCallback<BackendlessCollection<Contact>>(
  collection =>
  {
  },
  fault =>
  {
  });
Backendless.Persistence.Of<Contact>().Find(query, findCallback);
```

- 特定 phonebook で age が指定範囲内のものを検索。

```
// use a sample above where PhoneBook is created with multiple contacts,
// then use the savedPhoneBook instance:

BackendlessDataQuery query = new BackendlessDataQuery();
query.WhereClause = "PhoneBook[Contacts].objectId='" + savedPhoneBook.ObjectId;
query.WhereClause += "' and ";
query.WhereClause += "Age >= 21 and Age <= 30";

AsyncCallback<BackendlessCollection<Contact>> findCallback = new
    AsyncCallback<BackendlessCollection<Contact>>(
  collection =>
  {
  },
  fault =>
  {
  });
Backendless.Persistence.Of<Contact>().Find(query, findCallback);
```

- 特定 phonebook で city がTokyo、age が20以上のものを検索。

```
// use a sample above where PhoneBook is created with multiple contacts,
// then use the savedPhoneBook instance:

BackendlessDataQuery query = new BackendlessDataQuery();
```

```
query.WhereClause = "PhoneBook[Contacts].objectId='" + savedPhoneBook.ObjectId;
query.WhereClause += "' and ";
query.WhereClause += "Age > 20 and Address.City = 'Tokyo'";

AsyncCallback<BackendlessCollection<Contact>> findCallback = new
    AsyncCallback<BackendlessCollection<Contact>>(
  collection =>
  {
  },
  fault =>
  {
  });
Backendless.Persistence.Of<Contact>().Find(query, findCallback);
```

- 過去にUpdateしたことのあるものを検索。

```
// use a sample above where PhoneBook is created with multiple contacts,
// then use the savedPhoneBook instance:

DateTime now = DateTime.Now;
long timestamp = System.Convert.ToInt64(((DateTime)now - new DateTime(1970, 1, 1, 0, 0,
    0)).TotalMilliseconds);

savedPhoneBook.Contacts[0].Phone = "999-999-999";

AsyncCallback<PhoneBook> updateCallback = new AsyncCallback<PhoneBook>(
  updatedPhoneBook =>
  {
    BackendlessDataQuery query = new BackendlessDataQuery();
    query.WhereClause = "PhoneBook[Contacts].objectId='" + savedPhoneBook.ObjectId;
    query.WhereClause += "' and ";
    query.WhereClause += "updated > " + timestamp;

    AsyncCallback<BackendlessCollection<Contact>> findCallback = new
    AsyncCallback<BackendlessCollection<Contact>>(
      collection =>
      {
      },
      fault =>
      {
      });
    Backendless.Persistence.Of<Contact>().Find(query, findCallback);
  },
  fault =>
  {
    System.Diagnostics.Trace.WriteLine("ERROR " + fault);
  });

Backendless.Persistence.Of<PhoneBook>().Save(savedPhoneBook, updateCallback);
```

3 File Service

Backendless のファイルサービスはファイルストレージに対する操作APIを提供します。ファイルに対するアクセスコントロールを与えることで特定ユーザや特定グループだけ閲覧／改変できるような制御が可能です。

アップロードされたファイルはBackendlessのコンソールで確認できるだけでなく、ブラウザ経由でのアクセしが可能です。

3.1. File Upload

♦ **Asynchronousの呼出関数:**

```
public void Upload( Stream stream, string remotePath, AsyncCallback<BackendlessFile>
    callback );
```

```
public void Upload(Stream stream, string remotePath, UploadCallback uploadCallback,
    AsyncCallback<BackendlessFile> callback)
```

♦ **Parameter:**

stream - Backendless file storageにアップロードするストリームを指定します。Stream は FileStream, MemoryStreamのようにStream クラスを承継されたクラスを使用することができます。

remotePath – ファイルが格納されるBackendless file storageのパスとファイル名を指定します。

uploadCallback - アップロードの進捗状況を取得するコールバック関数を指定します。.

callback - Asynchronous関数の結果を取得するコールバック関数を指定します.

Asynchronousの呼出例:

```
AsyncCallback<BackendlessFile> callback = new AsyncCallback<BackendlessFile>(
  result =>
  {
    Debug.Log("File uploaded. URL - " + result.FileURL);
  },

  fault =>
  {
    Debug.Log("Error - " + fault);
  });

TextAsset asset = Resources.Load("backendless_png") as TextAsset;
Stream assetStream = new MemoryStream(asset.bytes);
BackendlessAPI.Backendless.Files.Upload(assetStream, "myfiles/backendless.png", callback);

byte[] byteArray = System.Text.Encoding.ASCII.GetBytes("Hello world");
Stream stream = new MemoryStream(byteArray);
BackendlessAPI.Backendless.Files.Upload(stream, "myfiles/hello.txt", callback);
```

3.2. File Deletion

- **Asynchronousの呼出関数:**

```
public void Remove( string fileUrl, AsyncCallback<object> callback );
```

- **Synchronousの呼出関数:**

```
public void Remove( string fileUrl );
```

- **Parameter:**

fileUrl – 削除するファイルのパスを指定します。

callback - Asynchronous関数の結果を取得するコールバック関数を指定します。

- **Asynchronousの呼出例:**

```
AsyncCallback<object> callback = new AsyncCallback<object>(
  result =>
  {
    Debug.Log("File deleted");
  },

  fault =>
  {
    Debug.Log("Error - " + fault);
  });

BackendlessAPI.Backendless.Files.Remove("myfiles/test.txt", callback);
```

- **Synchronousの呼出例:**

```
try
{
  BackendlessAPI.Backendless.Files.Remove("myfiles/test.txt");
}
catch (BackendlessException e)
{
  Debug.Log("Error - " + e);
}
```

3.3. Directory Deletion

- **Asynchronousの呼出関数:**

```
public void RemoveDirectory( string directoryPath, AsyncCallback<object> callback );
```

- **Synchronousの呼出関数:**

```
public void RemoveDirectory( string directoryPath );
```

♦ **Parameter:**

directoryPath - 削除するディレクトリのパスを指定します.

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します.

♦ **Asynchronousの呼出例:**

```
AsyncCallback<object> callback = new AsyncCallback<object>(
  result =>
  {
    Debug.Log("Directory deleted");
  },

  fault =>
  {
    Debug.Log("Error - " + fault);
  });

BackendlessAPI.Backendless.Files.RemoveDirectory("myfiles/pics", callback);
```

♦ **Synchronousの呼出例:**

```
try
{
  BackendlessAPI.Backendless.Files.RemoveDirectory("myfiles/pics");
}
catch (BackendlessException e)
{
  Debug.Log("Error - " + e);
}
```

4 Messaging Service

BackendlessのMessaging Service は「チャンネル」に対して発言・購読を指定することでメッセージングボード機能を提供します。またPush通知機能も本サービスに分類されます。

4.1. Push Notifications

♦ **Backendless ApplicationのPush Notificationsの設定:**

Push通知機能を利用するためには、Backendlessコンソール上でPush通知のための情報を設定する必要があります。詳しくは以下をご確認下さい。

iOS 設定・・・ http://backendless.com/documentation/messaging/ios/messaging_push_notification_setup_ios.htm

Android設定・・・http://backendless.com/documentation/messaging/ios/messaging_push_notification_setup_androi.htm

♦ **Push通知ためのUnity Plugin 設定:**

Backendless_unity3d_SDK_start-up ドキュメントにしたがって、backendless.unitypackageを追加してください。このとき、backendless.unitypackageのPlugins folder下位ファイルのインポートも必須になりますのでチェックボックスを外さずにまるごとインポートしてください。

Push Pluginを利用するためには、BackendlessPlugin.csにENABLE_PUSH_PLUGIN defineをしてください。（Pluginsを必要としない場合にPush Plugin によるコードでワーニングが発生しないよう、デフォルトでは、ENABLE_PUSH_PLUGINはコメントアウトされています。）

//#define ENABLE_PUSH_PLUGIN => #define ENABLE_PUSH_PLUGIN

♦ **Android Push Notifications:**

Android Push Notificationsのために下記のファイルが用意されています。

UNITY_PROJECT/Assets/Plugins/Android/AndroidManifest_backendless.xml

UNITY_PROJECT/Assets/Plugins/Android/BackendlessPushPlugin.jar

● マニフェストの編集

AndroidManifest_backendless.xmlは AndroidManifest.xml のサンプルです。この中の、YOUR_APPLICATION_PACKAGE_NAMEのところを実際に使用されるAndroidパッケージ名に変更してください。androidパッケージ名はunity projectのandroid build settingsに設定された”Bundle Identifier"と一致しなければなりません。

Unity projectでBackendless Unty3d SDKのAndroid Pluginを単独で使用する場合は、ファイル名をAndroidManifest_backendless.xmlからAndroidManifest.xmlに変更することでそのままご利用いただけます。

他のAndroid PluginとBackendless Unty3d SDKのAndroid Pluginを一緒にしようする場合は、AndroidManifest_backendless.xmlに記載されたGCMのための設定を、ご利用されるAndroidManifest.xmlに含まれるように追加してください。

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>

<uses-permission
      android:name="YOUR_APPLICATION_PACKAGE_NAME.permission.C2D_MESSAGE"/>
<permission
      android:name="YOUR_APPLICATION_PACKAGE_NAME.permission.C2D_MESSAGE"
      android:protectionLevel="signature"/>

<application android:icon="@drawable/app_icon" android:label="@string/app_name">

   <activity android:name="com.backendless.push.BackendlessUnityPlayerActivity"
    android:label="@string/app_name" android:launchMode="singleTask"
    android:configChanges="fontScale|keyboard|keyboardHidden|locale|mnc|mcc|navigation|orienta
    tion|screenLayout|screenSize|smallestScreenSize|uiMode|touchscreen">
      <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
   </activity>

   <receiver
         android:name="com.backendless.push.PushReceiver"
         android:permission="com.google.android.c2dm.permission.SEND">
      <intent-filter>
         <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
         <action android:name="com.google.android.c2dm.intent.REGISTRATION"/>

         <category android:name="YOUR_APPLICATION_PACKAGE_NAME"/>
      </intent-filter>
   </receiver>
</application>
```

♦ **iOS Push Notifications:**

iOS Push Notificationsのために下記のファイルが用意されています。

UNITY_PROJECT/Assets/Plugins/iOS/UIApplication_backendless.h

UNITY_PROJECT/Assets/Plugins/iOS/UIApplication_backendless.m

ファイル名のソースコードを修正することなく、そのまま使用して頂いても大丈夫です。

原則他のPluginと同時利用頂いても問題無いようにしていますが、問題が発生した場合は、Application_backendless.m の実行に影響を与えないような形で同時に使用するPlugin codeを修正してください。

4.2. Device Registration

デバイスをAPNSもしくはGCMに登録します。

♦ **Asynchronousの呼出関数:**

```
public void RegisterDevice(string GCMSenderID, AsyncCallback<string> callback = null);

public void RegisterDevice(string GCMSenderID, string channel, AsyncCallback<string> callback
    = null);

public void RegisterDevice(string GCMSenderID, List<string> channels, AsyncCallback<string>
    callback = null);

public void RegisterDevice(string GCMSenderID, DateTime expiration, AsyncCallback<string>
    callback = null);

public void RegisterDevice(string GCMSenderID, List<string> channels, DateTime? expiration,
    AsyncCallback<string> callback = null);
```

♦ **Synchronousの呼出関数:**

RegisterDevice apiはSynchronous関数をサポートしません。

♦ **Parameter:**

GCMSenderID - Android deviceの場合にはGoogle application project IDを指定します。 iOSの場合は空文字列 ""を指定してください。

channel(s) – deviceが所属するchannelを指定します。channel についての詳細な説明はBackendlessのドキュメントを参照して下さい。

expiration – deviceの登録が期限切れとなるtimestampを指定します。

callback - Asynchronous関数の結果を取得するcallback関数を指定します。

♦ **Asynchronousの呼出例:**

```
#if UNITY_ANDROID || UNITY_IPHONE
AsyncCallback<string> callback = new AsyncCallback<string>(
  registrationId =>
  {
  },
  fault =>
  {
  });
#endif

#if UNITY_ANDROID
string GCMSenderID = "000000000000";
Backendless.Messaging.RegisterDevice(GCMSenderID, callback);
#elif UNITY_IPHONE
Backendless.Messaging.RegisterDevice("", callback);
#endif
```

4.3. Retrieve Device Registration

♦ **Asynchronous の呼出関数:**

```
public void GetRegistrations( AsyncCallback<List<DeviceRegistration>> callback );
```

♦ **Synchronousの呼出関数:**

```
public List<DeviceRegistration> GetRegistrations();
```

♦ **Parameter:**

callback - Asynchronous 関数の結果を取得するcallback関数を指定します。

♦ **Asynchronousの呼出例:**

```
AsyncCallback<List<DeviceRegistration>> callback = new
    AsyncCallback<List<DeviceRegistration>>(
  registrations =>
  {
  },
  fault =>
  {
  });
Backendless.Messaging.GetRegistrations(callback);
```

♦ **Synchronousの呼出例:**

```
try
{
  List<DeviceRegistration> registrations = Backendless.Messaging.GetRegistrations();
}
catch(BackendlessException e)
{
}
```

4.4. Cancel Device Registration

♦ **Asynchronousの呼出関数:**

```
public void UnregisterDevice( AsyncCallback<bool> callback );
```

♦ **Synchronousの呼出関数:**

UnregisterDevice apiはSynchronous関数をサポートしません。

♦ **Asynchronousの呼出例:**

```
AsyncCallback<bool> callback = new AsyncCallback<bool>(
  result =>
  {
```

```
  },
  fault =>
  {
  });
Backendless.Messaging.UnregisterDevice(callback);
```

4.5. Message Publishing

- **Asynchronousの呼出関数:**

```
public void Publish( object message, AsyncCallback<MessageStatus> callback );
public void Publish( object message, string channelName, AsyncCallback<MessageStatus>
    callback );
```

```
public void Publish( object message, PublishOptions publishOptions,
    AsyncCallback<MessageStatus> callback );
public void Publish( object message, string channelName, PublishOptions publishOptions,
    AsyncCallback<MessageStatus> callback );
```

```
public void Publish( object message, DeliveryOptions deliveryOptions,
    AsyncCallback<MessageStatus> callback );
public void Publish( object message, string channelName, DeliveryOptions deliveryOptions,
    AsyncCallback<MessageStatus> callback );
```

```
public void Publish( object message, PublishOptions publishOptions, DeliveryOptions
    deliveryOptions, AsyncCallback<MessageStatus> callback );
public void Publish( object message, string channelName, PublishOptions publishOptions,
    DeliveryOptions deliveryOptions, AsyncCallback<MessageStatus> callback );
```

- **Synchronousの呼出関数:**

```
public MessageStatus Publish( object message );
public MessageStatus Publish( object message, string channelName );
```

```
public MessageStatus Publish( object message, PublishOptions publishOptions );
public MessageStatus Publish( object message, string channelName, PublishOptions
    publishOptions );
```

```
public MessageStatus Publish( object message, DeliveryOptions deliveryOptions );
public MessageStatus Publish( object message, string channelName, DeliveryOptions
    deliveryOptions );
```

```
public MessageStatus Publish( object message, PublishOptions publishOptions,
    Messaging.DeliveryOptions deliveryOptions );
public MessageStatus Publish( object message, string channelName, PublishOptions
    publishOptions, DeliveryOptions deliveryOptions );
```

- **Parameter:**

message – publishするmessageを指定します。

channelName – messageを送信するchannelを指定します。

publishOptions - publish optionを設定したPublishOptions クラスのオブジェクトを指定します。

deliveryOptions - delivery optionを設定したDeliveryOptions クラスのオブジェクトを指定します。

callback - Asynchronous関数の結果を取得するcallback関数を指定します。

◆ 例

メッセージ発行

```
AsyncCallback<MessageStatus> callback = new AsyncCallback<MessageStatus>(
  status =>
  {
  },
  fault =>
  {
  });

Backendless.Messaging.Publish("hello world!", callback);
```

ヘッダ付きメッセージ発行

```
PublishOptions publishOptions = new PublishOptions();
publishOptions.Headers = new Dictionary<string, string>();
publishOptions.Headers.Add("city", "Tokyo");

AsyncCallback<MessageStatus> callback = new AsyncCallback<MessageStatus>(
  status =>
  {
  },
  fault =>
  {
  });

Backendless.Messaging.Publish("hello world!", publishOptions, callback);
```

特定サブトピックへの発行

```
PublishOptions publishOptions = new PublishOptions();
publishOptions.Subtopic = "news.business.newyork";

AsyncCallback<MessageStatus> callback = new AsyncCallback<MessageStatus>(
  status =>
  {
  },
  fault =>
  {
  });

Backendless.Messaging.Publish("get free coffee at Moonbucks today", publishOptions, callback);
```

PushNotificationのみに発行:

```
DeliveryOptions deliveryOptions = new DeliveryOptions();
deliveryOptions.PushPolicy = PushPolicyEnum.ONLY;
```

```
deliveryOptions.PushBroadcast = DeliveryOptions.ALL;

AsyncCallback<MessageStatus> callback = new AsyncCallback<MessageStatus>(
  status =>
  {
  },
  fault =>
  {
  });

Backendless.Messaging.Publish("Hi Devices!", deliveryOptions, callback);
```

特定デバイス種別（Androidのみ）を指定したうえでPush通知としてメッセージ送信:

```
DeliveryOptions deliveryOptions = new DeliveryOptions();
deliveryOptions.PushPolicy = PushPolicyEnum.ONLY;
deliveryOptions.PushBroadcast = DeliveryOptions.ANDROID;
PublishOptions publishOptions = new PublishOptions();
publishOptions.AddHeader(PublishOptions.ANDROID_TICKER_TEXT_TAG, "You just got a push
    notification!");
publishOptions.AddHeader(PublishOptions.ANDROID_CONTENT_TITLE_TAG, "This is a notification
    title");
publishOptions.AddHeader(PublishOptions.ANDROID_CONTENT_TEXT_TAG, "Push Notifications are
    cool");

AsyncCallback<MessageStatus> callback = new AsyncCallback<MessageStatus>(
  status =>
  {
  },
  fault =>
  {
  });

Backendless.Messaging.Publish("Hi Devices!", publishOptions, deliveryOptions, callback);
```

デバイスIDを指定してPush通知:

```
DeliveryOptions deliveryOptions = new DeliveryOptions();
deliveryOptions.PushPolicy = PushPolicyEnum.ONLY;
deliveryOptions.PushSinglecast = new List<string>();
deliveryOptions.PushSinglecast.Add("receiver-device-id");
deliveryOptions.PushBroadcast = DeliveryOptions.ANDROID;
PublishOptions publishOptions = new PublishOptions();
publishOptions.AddHeader(PublishOptions.ANDROID_TICKER_TEXT_TAG, "You just got a push
    notification!");
publishOptions.AddHeader(PublishOptions.ANDROID_CONTENT_TITLE_TAG, "This is a notification
    title");
publishOptions.AddHeader(PublishOptions.ANDROID_CONTENT_TEXT_TAG, "Push Notifications are
    cool");

AsyncCallback<MessageStatus> callback = new AsyncCallback<MessageStatus>(
  status =>
  {
  },
  fault =>
  {
  });

Backendless.Messaging.Publish(("This is a private message!", publishOptions, deliveryOptions,
    callback);
```

20秒間遅延させて送信

```
DeliveryOptions deliveryOptions = new DeliveryOptions();
```

```
deliveryOptions.RepeatExpiresAt = DateTime.Now.AddSeconds(20);

AsyncCallback<MessageStatus> callback = new AsyncCallback<MessageStatus>(
  status =>
  {
  },
  fault =>
  {
  });

Backendless.Messaging.Publish("This message was scheduled 20 sec ago", deliveryOptions,
    callback);
```

指定間隔で定期的に通知:

```
DeliveryOptions deliveryOptions = new DeliveryOptions();
deliveryOptions.RepeatExpiresAt = DateTime.Now.AddSeconds(60);
deliveryOptions.RepeatEvery = 20; // the message will be delivered every 20 seconds

AsyncCallback<MessageStatus> callback = new AsyncCallback<MessageStatus>(
  status =>
  {
  },
  fault =>
  {
  });

Backendless.Messaging.Publish("This message was scheduled 20 sec ago", deliveryOptions,
    callback);
```

4.6. Publish Push Notifications

Pushメッセージの発行時に、message headerを指定します。 Message header はGCMやAPNS等異なるサービス毎に必要な情報が記載された、Jsonオブジェクトとして記述します。

下記のheaderをpublish options objectに追加して使用できます。

Operating System	Headers	Description
iOS	"ios-alert":value	iOS 用のアラートメッセージの文字列を設定します。 Headerが存在しない状態でiOSデバイスを対象として 通知が発行された場合、Backendlessは"message" argumentの値をheaderとして設定します。 これを避ける場合ios-alert headerをnullに設定してください。
	"ios-badge":value	Badgeを更新する値を設定します。
	"ios-sound":URL string or array of bytes	サウンド通知から再生するサウンドのバイト配列またはURLを設定します。
Android	"android-ticker-text":value	Android用のメッセージです。デバイスに通知がとどいたタイミングで、デバイス画面の上の方にあるtickerテキストを設定します。
	"android-content-title":value	Android Notification Centerに表示されるタイトルを設定します。
	"android-content-text":value	Android Notification Centerでandroid-content-titleの下に表示される通知メッ

	セージを設定します。

プッシュ通知Backendless consoleから直接発行したり、APIを使用することができます。(Message Publishingセクションの例をご参考ください。)

これらの通知はBackendlessコンソール上でJSON形式の文字列を指定することで端末に送信することができますが、別途ウェブアプリ上で Message Publishing のAPIを利用することで運営担当者専用にお好みのコンソール画面を用意し自作頂くこともできます。

4.7. Cancel Scheduled Message

♦ **Asynchronousの呼出関数:**

```
public void Cancel( string messageId, AsyncCallback<bool> callback );
```

♦ **Synchronousの呼出関数:**

```
public bool Cancel( string messageId );
```

♦ **Parameter:**

messageId - キャンセルするmessageのIDを指定します。

callback - Asynchronous関数の結果を取得するcallback関数を指定します。

Example:

```
DeliveryOptions deliveryOptions = new DeliveryOptions();
deliveryOptions.RepeatExpiresAt = DateTime.Now.AddSeconds(20);

AsyncCallback<MessageStatus> callback = new AsyncCallback<MessageStatus>(
  status =>
  {
    AsyncCallback<bool> cancelCallback = new AsyncCallback<bool>(
      cancelled =>
      {
      },
      fault =>
      {
      });
    Backendless.Messaging.Cancel(status.MessageId, cancelCallback);
  },
  fault =>
  {
  });

Backendless.Messaging.Publish("Test Message", deliveryOptions, callback);
```

4.8. Message Subscription

♦ **Asynchronous の呼出関数:**

```
public void Subscribe( AsyncCallback<List<Message>> callback, AsyncCallback<Subscription>
    subscriptionCallback );
public void Subscribe( string channelName, AsyncCallback<List<Message>> callback,
    AsyncCallback<Subscription> subscriptionCallback );
```

```
public void Subscribe( int pollingInterval, AsyncCallback<List<Message>> callback,
    AsyncCallback<Subscription> subscriptionCallback );
public void Subscribe( string channelName, int pollingInterval, AsyncCallback<List<Message>>
    callback, AsyncCallback<Subscription> subscriptionCallback );
```

```
public void Subscribe( AsyncCallback<List<Message>> callback, Messaging.SubscriptionOptions
    subscriptionOptions, AsyncCallback<Subscription> subscriptionCallback );
public void Subscribe( string channelName, AsyncCallback<List<Message>> callback,
    Messaging.SubscriptionOptions subscriptionOptions, AsyncCallback<Subscription>
    subscriptionCallback );
```

```
public void Subscribe( int pollingInterval, AsyncCallback<List<Message>> callback,
    Messaging.SubscriptionOptions subscriptionOptions, AsyncCallback<Subscription>
    subscriptionCallback );
public void Subscribe( string channelName, int pollingInterval, AsyncCallback<List<Message>>
    callback, Messaging.SubscriptionOptions subscriptionOptions, AsyncCallback<Subscription>
    subscriptionCallback );
```

♦ **Synchronous の呼出関数:**

```
public Messaging.Subscription Subscribe( AsyncCallback<List<Message>> callback );
public Messaging.Subscription Subscribe( string channelName, AsyncCallback<List<Message>>
    callback );
```

```
public Messaging.Subscription Subscribe( int pollingInterval, AsyncCallback<List<Message>>
    callback );
public Messaging.Subscription Subscribe( string channelName, int pollingInterval,
    AsyncCallback<List<Message>> callback );
```

```
public Messaging.Subscription Subscribe( AsyncCallback<List<Message>> callback,
    Messaging.SubscriptionOptions subscriptionOptions );
public Messaging.Subscription Subscribe( string channelName, AsyncCallback<List<Message>>
    callback, Messaging.SubscriptionOptions subscriptionOptions );
```

```
public Messaging.Subscription Subscribe( int pollingInterval, AsyncCallback<List<Message>>
    callback, Messaging.SubscriptionOptions subscriptionOptions );
public Messaging.Subscription Subscribe( string channelName, int pollingInterval,
    AsyncCallback<List<Message>> callback, Messaging.SubscriptionOptions subscriptionOptions )
```

♦ **Parameter:**

channelName - messageを受信するchannelを指定します。

subscriptionResponder - messageを受信するcallback関数を指定します。

subscriptionOptions - subscription optionを設定したSubscriptionOptions class variableを指定します。

callback - Asynchronous関数の結果を取得するcallback関数を指定します。

- **Example:**

```
Subscription mySubscription = null;
AsyncCallback<List<Message>> callback = new AsyncCallback<List<Message>>(
  messages =>
  {
    foreach (Message message in messages)
    {
      Debug.Log("PublisherId=" + message.PublisherId);
      Debug.Log("Data=" + message.Data);
    }
  },
  fault =>
  {
  });

AsyncCallback<Subscription> subscriptionCallback = new AsyncCallback<Subscription>(
  subscription =>
  {
    mySubscription = subscription;
  },
  fault =>
  {
  });

Backendless.Messaging.Subscribe(channelName, callback, subscriptionCallback);
```

4.9. Cancel Subscription

- **メソッド（Sync のみ）：**

```
subscriptionObject. CancelSubscription();
```

- **Example:**

```
Subscription mySubscription = null;
AsyncCallback<List<Message>> callback = new AsyncCallback<List<Message>>(
  messages =>
  {
    foreach (Message message in messages)
    {
      Debug.Log("PublisherId=" + message.PublisherId);
      Debug.Log("Data=" + message.Data);
    }
  },
  fault =>
  {
  });

AsyncCallback<Subscription> subscriptionCallback = new AsyncCallback<Subscription>(
  subscription =>
  {
    subscription.CancelSubscription();
  },
  fault =>
  {
```

```
  });

Backendless.Messaging.Subscribe(channelName, callback, subscriptionCallback);
```

4.10. Sending Email

BackendlessのMessaging Service はメール送信にも対応しています。送信するSMTPサーバはコンソール画面で設定頂くことが可能です。

- **Asynchronous の呼出関数:**

```
public void SendTextEmail(String subject, String messageBody, List<String> recipients,
    AsyncCallback<object> responder);
public void SendTextEmail(String subject, String messageBody, String recipient,
    AsyncCallback<object> responder);
```

```
public void SendHTMLEmail(String subject, String messageBody, List<String> recipients,
    AsyncCallback<object> responder);
public void SendHTMLEmail(String subject, String messageBody, String recipient,
    AsyncCallback<object> responder);
```

```
public void SendEmail(String subject, BodyParts bodyParts, String recipient, List<String>
    attachments, AsyncCallback<object> responder);
public void SendEmail(String subject, BodyParts bodyParts, String recipient,
    AsyncCallback<object> responder);
public void SendEmail(String subject, BodyParts bodyParts, List<String> recipients,
    List<String> attachments, AsyncCallback<object> responder);
```

- **Synchronous の呼出関数:**

```
public void SendTextEmail(String subject, String messageBody, List<String> recipients);
public void SendTextEmail(String subject, String messageBody, String recipient)
```

```
public void SendHTMLEmail(String subject, String messageBody, List<String> recipients);
public void SendHTMLEmail(String subject, String messageBody, String recipient);
```

```
public void SendEmail(String subject, BodyParts bodyParts, String recipient, List<String>
    attachments);
public void SendEmail(String subject, BodyParts bodyParts, String recipient);
public void SendEmail(String subject, BodyParts bodyParts, List<String> recipients,
    List<String> attachments)
```

- **Parameter:**

subject - Email messageのsubjectを指定します。

messageBody - Email messageのTextまたはHTML bodyを指定します。

bodyParts - TextとHTMLのmessagebodyを含むcom.backendless.messaging.BodyParts class instanceを指定します。

recipient - Email messageを伝達するEmail addressを指定します。

recipients - Email messageを伝達するEmail addressリストを指定します。

attachments - Backendless File Serviceのファイル経路リストを指定します。参照されたファイルはEmail messageに添付されます。

responder - Asynchronous関数の結果を受けるcallback関数を指定します。

♦ **Example:**

```
AsyncCallback<object> responder = new AsyncCallback<object>(
  result =>
  {
    System.Console.WriteLine( "[ASYNC] message sent" );
  },

  fault =>
  {
    System.Console.WriteLine( "Error - " + fault );
  } );

// async request. Plain text message to one recipient
Backendless.Messaging.SendTextEmail( "Reminder", "Hey JB! Your car will be ready by 5pm",
    "james.bond@mi6.co.uk", responder );

// sync request. HTML messahe to multiple recipients
List<String> recipients = new List<String>();
recipients.Add( "mom@gmail.com" );
recipients.Add( "dad@gmail.com" );

String mailBody = "Guys, the dinner last night was <b>awesome</b>";

Backendless.Messaging.SendHTMLEmail( "Dinner", mailBody, recipients );

System.Console.WriteLine( "[SYNC] email has been sent" );
```

5 Geolocation Service

Geolocation Service は緯度経度とメタ情報を保持する “Geo Point” を管理・検索するサービスです。

5.1. Adding a Geo Category

Geo Category はGeo Pointを保持するテーブルです。

- **Asynchronousの呼出関数:**

```
public GeoCategory AddCategory(string categoryName);
```

Synchronousの呼出関数:

```
public void AddCategory(string categoryName, AsyncCallback<GeoCategory> callback);
```

- **Parameter:**

categoryName - 追加するカテゴリ名を指定します。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

- **Asynchronousの呼出例:**

```
AsyncCallback<GeoCategory> callback = new AsyncCallback<GeoCategory>(
  result =>
  {
  Debug.Log("Category created - " + result.Name);
  },
  fault =>
  {
  Debug.Log("Error - " + fault);
  });

Backendless.Geo.AddCategory("my_category", callback);
```

- **Synchronousの呼出例:**

```
try
{
  GeoCategory category = Backendless.Geo.AddCategory("my_category");
  Debug.Log("Category created - " + category.Name);
}
catch (BackendlessException e)
{
  Debug.Log("Error - " + e);
}
```

5.2. Deleting a Geo Category

Geo Category を削除します。

♦ **Asynchronousの呼出関数:**

```
public void DeleteCategory(string categoryName, AsyncCallback<bool> callback);
```

♦ **Synchronousの呼出関数:**

```
public bool DeleteCategory(string categoryName);
```

♦ **Parameter:**

categoryName - 削除するカテゴリ名を指定します。

callback - Asynchronous関数の結果を取得するコールバック関数を指定します。

♦ **Asynchronousの呼出例:**

```
AsyncCallback<bool> callback = new AsyncCallback<bool>(
  result =>
  {
    Debug.Log("Category deleted - " + result);
  },
  fault =>
  {
    Debug.Log("Error - " + fault);
  });

Backendless.Geo.DeleteCategory("my_category", callback);
```

♦ **Synchronousの呼出例:**

```
try
{
  bool result = Backendless.Geo.DeleteCategory("my_category");
  Debug.Log("Category deleted - " + result);
}
catch (BackendlessException e)
{
  Debug.Log("Error - " + e);
}
```

5.3. Retrieving Geo Categories

Geo Category を取得します。

♦ **Asynchronousの呼出関数:**

```
public List<GeoCategory> GetCategories();
```

♦ **Synchronousの呼出関数:**

```
public void GetCategories(AsyncCallback<List<GeoCategory>> callback);
```

♦ **Parameter:**

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **Asynchronousの呼出例:**

```
AsyncCallback<List<GeoCategory>> callback = new AsyncCallback<List<GeoCategory>>(
  result =>
  {
    Debug.Log("Categories received - " + result.Count);
  },
  fault =>
  {
    Debug.Log("Error - " + fault);
});

Backendless.Geo.GetCategories(callback);
```

♦ **Synchronousの呼出例:**

```
try
{
  List<GeoCategory> categories = Backendless.Geo.GetCategories();
  Debug.Log("Categories received - " + categories.Count);
}
catch (BackendlessException e)
{
  Debug.Log("Error - " + e);
}
```

5.4. Adding a Geo Point

Geo Category に対して Geo Point を追加します。

♦ **Asynchronousの呼出関数:**

```
public void SavePoint(GeoPoint geoPoint, AsyncCallback<GeoPoint> callback);
public void SavePoint(double latitude, double longitude, Dictionary<string, string> metadata,
    AsyncCallback<GeoPoint> callback);
public void SavePoint(double latitude, double longitude, Dictionary<string, object> metadata,
    AsyncCallback<GeoPoint> callback);
public void SavePoint(double latitude, double longitude, List<string> categoryNames,
    Dictionary<string, string> metadata, AsyncCallback<GeoPoint> callback);
public void SavePoint(double latitude, double longitude, List<string> categoryNames,
    Dictionary<string, object> metadata, AsyncCallback<GeoPoint> callback);
```

♦ **Synchronousの呼出関数:**

```
public GeoPoint SavePoint(GeoPoint geoPoint);
public GeoPoint SavePoint(double latitude, double longitude, Dictionary<string, string>
    metadata);
```

```
public GeoPoint SavePoint(double latitude, double longitude, Dictionary<string, object>
    metadata);
public GeoPoint SavePoint(double latitude, double longitude, List<string> categoryNames,
    Dictionary<string, string> metadata);
public GeoPoint SavePoint(double latitude, double longitude, List<string> categoryNames,
    Dictionary<string, object> metadata);
```

- **Parameter:**

geoPoint - 追加するGeoPointclassを指定します。

latitude - 追加する緯度の座標を指定します。

longitude - 追加する経度の座標を指定します。

categoryNames - 位置を追加するカテゴリを指定します。

metadata - 位置のmetadataを指定します。

callback - Asynchronous関数の結果を取得するコールバック関数を指定します。

- **Asynchronousの呼出例:**

```
AsyncCallback<GeoPoint> callback = new AsyncCallback<GeoPoint>(
  result =>
  {
    Debug.Log("Geo point saved - " + result.ObjectId);
  },

  fault =>
  {
    Debug.Log("Error - " + fault);
  });

List<string> categories = new List<string>();
categories.Add("restaurants");
categories.Add("cool_places");

Dictionary<string, string> metadata = new Dictionary<string, string>();
metadata.Add("name", "Eatzi's");
GeoPoint geoPoint = new GeoPoint(32.81, -96.80, categories, metadata);
```

- **Synchronousの呼出例:**

```
try
{
  List<string> categories = new List<string>();
  categories.Add("restaurants");
  categories.Add("cool_places");

  Dictionary<string, string> metadata = new Dictionary<string, string>();
  metadata.Add("name", "Eatzi's");
  GeoPoint geoPoint = new GeoPoint(32.81, -96.80, categories, metadata);

  GeoPoint savedPoint = Backendless.Geo.SavePoint(geoPoint);
  Debug.Log("Geo point saved - " + savedPoint.ObjectId);
}
catch (BackendlessException e)
{
```

```
  Debug.Log("Error - " + e);
}
```

5.5. Updating a Geo Point

UpdateはAdd APIと同じメソッドを利用します。

GeoPointのObjectIdの属性をすでに持っている場合にはupdate、そうではない場合にはaddになります。

5.6. Search in Category

Geo Category 内でGeo Point を検索します。クエリオブジェクトを作成して検索条件にします。

♦ **Asynchronousの呼出関数:**

```
public void GetPoints(BackendlessGeoQuery geoQuery,
    AsyncCallback<BackendlessCollection<GeoPoint>> callback);
```

♦ **Synchronousの呼出関数:**

```
public BackendlessCollection<GeoPoint> GetPoints(BackendlessGeoQuery geoQuery);
```

♦ **Parameter:**

geoQuery - 検索のためのBackendlessGeoQuery クラス変数を指定します。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **例**

- カテゴリ内検索:

```
AsyncCallback<BackendlessCollection<GeoPoint>> callback = new
    AsyncCallback<BackendlessCollection<GeoPoint>>(
  result =>
  {
    Debug.Log("Found geo points - " + result.Data.Count);
  },
  fault =>
  {
    Debug.Log("Error - " + fault);
  });

BackendlessGeoQuery geoQuery = new BackendlessGeoQuery();
geoQuery.Categories.Add("Restaurants");
Backendless.Geo.GetPoints(geoQuery, callback);
```

- メタデータを用いた検索:

```
AsyncCallback<BackendlessCollection<GeoPoint>> callback = new
    AsyncCallback<BackendlessCollection<GeoPoint>>(
  result =>
  {
```

```
    Debug.Log("Found geo points - " + result.Data.Count);
  },
  fault =>
  {
    Debug.Log("Error - " + fault);
  });

BackendlessGeoQuery geoQuery = new BackendlessGeoQuery();
geoQuery.Categories.Add("Restaurants");
geoQuery.Metadata.Add("Cuisine", "French");
geoQuery.Metadata.Add("Atmosphere", "Romantic");
```

- レスポンスにメタデータを要求する場合:

```
AsyncCallback<BackendlessCollection<GeoPoint>> callback = new
    AsyncCallback<BackendlessCollection<GeoPoint>>(
  result =>
  {
    Debug.Log("Found geo points - " + result.Data.Count);
  },
  fault =>
  {
    Debug.Log("Error - " + fault);
  });

BackendlessGeoQuery geoQuery = new BackendlessGeoQuery();
geoQuery.Categories.Add("Restaurants");
geoQuery.IncludeMeta = true;
Backendless.Geo.GetPoints(geoQuery, callback);
```

5.7. Search in Radius

中心点となる経度緯度と半径を指定して範囲内を円形検索をします。

♦ **Asynchronousの呼出関数:**

```
public void GetPoints(BackendlessGeoQuery geoQuery,
    AsyncCallback<BackendlessCollection<GeoPoint>> callback);
```

♦ **Synchronousの呼出関数:**

```
public BackendlessCollection<GeoPoint> GetPoints(BackendlessGeoQuery geoQuery);
```

♦ **Parameter:**

geoQuery - 検索のための BackendlessGeoQuery クラス変数を指定します。

callback - Asynchronous関数の結果を取得するコールバック関数を指定します。

♦ **例：**

- カテゴリ内で緯度経度＋半径を指定して検索:

```
AsyncCallback<BackendlessCollection<GeoPoint>> callback = new
    AsyncCallback<BackendlessCollection<GeoPoint>>(
  result =>
  {
```

```
    Debug.Log("Found geo points - " + result.Data.Count);
  },
  fault =>
  {
    Debug.Log("Error - " + fault);
  });

BackendlessGeoQuery geoQuery = new BackendlessGeoQuery();
geoQuery.Categories.Add("Restaurants");
geoQuery.Latitude = 41.48;
geoQuery.Longitude = 2.15;
geoQuery.Radius = 100000;
geoQuery.Units = BackendlessAPI.Geo.Units.METERS;

Backendless.Geo.GetPoints(geoQuery, callback);
```

- 条件にメタデータを追加:

```
AsyncCallback<BackendlessCollection<GeoPoint>> callback = new
    AsyncCallback<BackendlessCollection<GeoPoint>>(
result =>
{
  Debug.Log("Found geo points - " + result.Data.Count);
},
fault =>
{
  Debug.Log("Error - " + fault);
});

BackendlessGeoQuery geoQuery = new BackendlessGeoQuery();
geoQuery.Categories.Add("Restaurants");
geoQuery.Latitude = 41.48;
geoQuery.Longitude = 2.15;
geoQuery.Radius = 100000;
geoQuery.Units = BackendlessAPI.Geo.Units.METERS;

geoQuery.Metadata.Add("Cuisine", "French");
geoQuery.Metadata.Add("Atmosphere", "Romantic");

Backendless.Geo.GetPoints(geoQuery, callback);
```

5.8. Search in Rectangular Area

経度緯度を二点指定すると、二点を対角とする矩形領域に対して検索を実行します。

- **Asynchronousの呼出関数:**

```
public void GetPoints(BackendlessGeoQuery geoQuery,
    AsyncCallback<BackendlessCollection<GeoPoint>> callback);
```

- **Synchronousの呼出関数:**

```
public BackendlessCollection<GeoPoint> GetPoints(BackendlessGeoQuery geoQuery);
```

- **Parameter:**

geoQuery - 検索のためのBackendlessGeoQuery クラス変数を指定します。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **例：**

- カテゴリ内を矩形検索します:

```
AsyncCallback<BackendlessCollection<GeoPoint>> callback = new
    AsyncCallback<BackendlessCollection<GeoPoint>>(
  result =>
  {
    Debug.Log("Found geo points - " + result.Data.Count);
  },
  fault =>
  {
    Debug.Log("Error - " + fault);
  });

BackendlessGeoQuery geoQuery = new BackendlessGeoQuery();
geoQuery.Categories.Add("Restaurants");
geoQuery.SearchRectangle = new double[] { 32.78, -96.8, 25.79, -80.22 };

Backendless.Geo.GetPoints(geoQuery, callback);
```

- 矩形＋メタデータで検索します:

```
AsyncCallback<BackendlessCollection<GeoPoint>> callback = new
    AsyncCallback<BackendlessCollection<GeoPoint>>(
result =>
{
  Debug.Log("Found geo points - " + result.Data.Count);
},
fault =>
{
  Debug.Log("Error - " + fault);
});

BackendlessGeoQuery geoQuery = new BackendlessGeoQuery();
geoQuery.Categories.Add("Restaurants");
geoQuery.SearchRectangle = new double[] { 32.78, -96.8, 25.79, -80.22 };

geoQuery.Metadata.Add("Cuisine", "French");
geoQuery.Metadata.Add("Atmosphere", "Romantic");

Backendless.Geo.GetPoints(geoQuery, callback);
```

6 Custom Business Logic

6.1. Overview

Backendless Custom Business Logicはサーバ側に開発者が作成したコードを設置し、application backendの拡張やカスタマイズが出来るように設計されたシステムです。

Custom Business Logicを使用すれば、開発者は既存のBackendlessサービスの基本動作を再定義したり、完全に新しいロジックを導入する事ができます。

Custom Business Logic の拡張方法は処理の駆動タイミングによって以下の二つの種類に分類できます。

> **event handlers** –既存のAPIの呼び出しの「直前」または「直後」に実行されます。既存のBackendlessのサービスの動作を拡張したり追加するのに利用することができます。例えばランキングボード用のテーブルにデータを保存した直後に1位になっているユーザに検出してメッセージ通知を行うといった利用が可能です。
>
> **timers** – 指定されたスケジュールで動作を一回もしくは繰り返し実行します。定期的な実行処理が必要な場合に利用することができます。

これらについての詳しい内容は下記のウェブページもご参照下さい。

http://backendless.com/documentation/business-logic/java/bl_overview.htm

6.2. Custom Events

Custom Event は開発者がイベントを自由に定義し、event handler 型の Custom Business Logic APIを呼び出すことができる仕組みです。Custom Event を用いれば、開発者はカスタマイズしたビジネスロジックをサーバ上に実現することができます（※実行可能時間等に制限があります。）

以下の説明はCustom Event を呼び出すAPIを説明していますが、実際に利用する場合はサーバ側の実行コードとセットでご利用下さい。

- **Asynchronousの呼出関数:**

```
public void Dispatch( String eventName, IDictionary eventArgs, AsyncCallback<IDictionary>
    callback );
```

- **Synchronouの呼出関数:**

```
public IDictionary Dispatch( String eventName, IDictionary eventArgs );
```

♦ **Parameter:**

eventName – dispatchするeventのnameを指定します。

eventArgs - event handlerに伝達するeventArgsを指定します。

callback - Asynchronous関数の結果を取得するcallback関数を指定します。

♦ **Asynchronousの呼出例:**

```
Dictionary<string, object> eventArgs = new Dictionary<string, object>();
eventArgs.Add("weather", "sunny");
AsyncCallback<IDictionary> callback = new AsyncCallback<IDictionary>(
  result =>
  {
    Debug.Log("received result - " + result);
  },
  fault =>
  {
    Debug.Log("Error - " + fault);
  });
Backendless.Events.Dispatch("foo", eventArgs, callback);
```

♦ **Synchronousの呼出例:**

```
try
{
  Dictionary<string, string> eventArgs = new Dictionary<string, string>();
  eventArgs.Add("weather", "sunny");
  IDictionary result = Backendless.Events.Dispatch("foo", eventArgs);
  Debug.Log("received result - " + result); }
catch(BackendlessException e)
{
  Debug.Log(e);
}
```

7 Utilities - Caching API

Caching API はオンメモリで高速かつ一時的な保存領域をサーバ上で提供する機能です。データはKey-Valueのペアで保存可能です。

7.1. Putting data into cache

♦ **Asynchronousの呼出関数:**

```
public void Put( String key, Object obj, AsyncCallback<Object> callback );
public void Put( String key, Object obj, int expire, AsyncCallback<Object> callback );
```

♦ **Synchronousの呼出関数:**

```
public void Put( String key, Object obj );
public void Put( String key, Object obj, int expire );
```

♦ **Parameter:**

key - cacheを識別することができるobject keyを指定します。

obj - cacheに配置するobjectを指定します。

expire - objectを cacheに維持する時間を指定します(秒単位)。時間を指定するとBackendlessが自動的に削除します。。基本値は 7200秒です。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<Object> callback = new AsyncCallback<Object>(
result =>
{
  System.Console.WriteLine( "object has been placed into cache" );
},
fault =>
{
  System.Console.WriteLine( "Error - " + fault );
} );

// putting string into cache
Backendless.Cache.Put( "foo", "hello world", callback );

// putting complex object into cache
Order order = Backendless.Persistence.Of<Order>().FindFirst();
Backendless.Cache.Put( "firstorder", order );
```

7.2. Retrieving data from cache

♦ **Asynchronousの呼出関数:**

```
public void Get<T>( String key, AsyncCallback<T> callback );
```

♦ **Synchronousの呼出関数:**

```
public T Get<T>( String key );
```

♦ **Parameter:**

key –取得したいobjectの key文字列を指定します。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<Order> callback = new AsyncCallback<Order>(
result =>
{
  System.Console.WriteLine( "received order object from cache - "  + result.Name );
},
fault =>
{
  System.Console.WriteLine( "Error - " + fault );
} );

// get object from cache asynchronously
Backendless.Cache.Get<Order>( "firstorder", callback );

// get object from cache synchronously
Order order = Backendless.Cache.Get<Order>( "firstorder" );
```

7.3. Checking if key exists in cache

キャッシュ中に指定されたKeyのオブジェクトが残っているかをチェックします。

♦ **Asynchronousの呼出関数:**

```
public void Contains( String key, AsyncCallback<Boolean> callback );
```

♦ **Synchronousの呼出関数:**

```
public Boolean Contains( String key );
```

♦ **Parameter:**

key - cacheにobjectが存在するのか確認するためのobject keyを指定します。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<Boolean> callback = new AsyncCallback<Boolean>(
result =>
{
  System.Console.WriteLine( "[ASYNC] object exists in cache - "  + result );
},
fault =>
{
  System.Console.WriteLine( "Error - " + fault );
} );

// get object from cache asynchronously
Backendless.Cache.Contains( "firstorder", callback );

// get object from cache synchronously
Boolean objectExists = Backendless.Cache.Contains( "firstorder" );
System.Console.WriteLine( "[SYNC] object exists in cache - "  + objectExists );
```

7.4. Extending object's life in cache

キャッシュ内のオブジェクトの生存期間を延長します。

♦ **Asynchronousの呼出関数:**

```
public void ExpireIn( String key, int seconds, AsyncCallback<Object> callback );
public void ExpireAt( String key, int timestamp, AsyncCallback<Object> callback );
```

♦ **Synchronousの呼出関数:**

```
public void ExpireIn( String key, int seconds );
public void ExpireAt( String key, int timestamp );
```

♦ **Parameter:**

key -生存期間を延ばすobject keyを指定します。

seconds - objectの生存期間を延長する秒を指定します。範囲は、1～7200の間の値でなくてはなりません。

timestamp - objectが期限切れになるミリ秒単位のtimestampを指定します。指定範囲は、現在の時刻～7200000 ミリ秒の間の値である必要があります。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<Object> callback = new AsyncCallback<Object>(
result =>
{
  System.Console.WriteLine( "[ASYNC] object life has been extended" );
},
fault =>
{
```

```
  System.Console.WriteLine( "Error - " + fault );
} );

// extend object's life in cache asynchronously by 1200 seconds - 20 minutes
Backendless.Cache.ExpireIn( "firstorder", 1200, callback );

// extend object's life in cache synchronously by 1200 seconds - 20 minutes

long timestamp = DateTime.Now.Ticks / TimeSpan.TicksPerMillisecond + 72000000;
Backendless.Cache.ExpireAt( "firstorder", timestamp );
```

7.5. Deleting object from cache

- **Asynchronousの呼出関数:**

```
public void Delete( String key, AsyncCallback<Object> callback );
```

- **Synchronousの呼出関数:**

```
public void Delete( String key );
```

- **Parameter:**

key - cacheで削除するobject keyを指定します。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

- **呼出関数:**

```
AsyncCallback<Object> callback = new AsyncCallback<Object>(
result =>
{
  System.Console.WriteLine( "[ASYNC] object has been deleted from cache" );
},
fault =>
{
  System.Console.WriteLine( "Error - " + fault );
} );

// delete object asynchronously
Backendless.Cache.Delete( "firstorder", callback );

// delete object synchronously
Backendless.Cache.Delete( "firstorder" );
```

8 Utilities - Atomic Counters API

サーバ側で動作するアトミックなカウンタを提供します。カウンタには名前（counterName）を指定して複数利用できます。APIのバリエーションとして１もしくはNの増減を指定するもの、戻り値として増減操作直前の値（previous）か現在(current)取得ができるものが用意されています。

8.1. Increment by 1, return previous

♦ **Asynchronousの呼出関数:**

```
// Backendless.Counters approach
public void Backendless.Counters.GetAndIncrement<T>( String counterName, AsyncCallback<T>
    callback );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public void counter.GetAndIncrement( AsyncCallback<T> callback );
```

♦ **Synchronousの呼出関数:**

```
// Backendless.Counters approach
public int Backendless.Counters.GetAndIncrement( String counterName );
public T Backendless.Counters.GetAndIncrement<T>( String counterName );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public T counter.GetAndIncrement();
```

♦ **Parameter:**

counterName - 増加するcounter 名前を指定します。

T - counter 値の数字形式を指定します。byte、 short、 int、 float、 long and doubleを指定することができます。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<Object> callback = new AsyncCallback<Object>(
result =>
{
  System.Console.WriteLine( "object has been placed into cache" );
},
fault =>
{
  System.Console.WriteLine( "Error - " + fault );
} );

// putting string into cache
Backendless.Cache.Put( "foo", "hello world", callback );

// putting complex object into cache
Order order = Backendless.Persistence.Of<Order>().FindFirst();
Backendless.Cache.Put( "firstorder", order );
```

8.2. Increment by 1, return current

♦ **Asynchronousの呼出関数:**

```
// Backendless.Counters approach
public void Backendless.Counters.IncrementAndGet<T>( String counterName, AsyncCallback<T>
    callback );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public void counter.IncrementAndGet( AsyncCallback<T> callback );
```

♦ **Synchronousの呼出関数:**

```
// Backendless.Counters approach
public int Backendless.Counters.IncrementAndGet( String counterName );
public T Backendless.Counters.IncrementAndGet<T>( String counterName );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public T counter.IncrementAndGet();
```

♦ **Parameter:**

counterName - 増加する counter 名前を指定します。

T - counter 値の数字形式を指定します。byte、 short、 int、 float、 long and doubleを指定することができます。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<int> callback = new AsyncCallback<int>(
  result =>
  {
    System.Console.WriteLine( "[ASYNC] counter value is - " + result );
  },
  fault =>
  {
     System.Console.WriteLine( "Error - " + fault );
  } );

Backendless.Counters.IncrementAndGet( "my counter", callback );

IAtomic<int> myCounter = Backendless.Counters.Of<int>( "my counter" );
int counter = myCounter.IncrementAndGet();
System.Console.WriteLine( "[SYNC] counter value through IAtomic is - " + counter );

counter = Backendless.Counters.IncrementAndGet( "my counter" );
System.Console.WriteLine( "[SYNC] counter value is - " + counter );

long longCounter = Backendless.Counters.IncrementAndGet<long>( "my counter" );
System.Console.WriteLine( "[SYNC] counter value is - " + longCounter );
```

8.3. Decrement by 1, return previous

♦ **Asynchronousの呼出関数:**

```
// Backendless.Counters approach
public void Backendless.Counters.GetAndDecrement<T>( String counterName, AsyncCallback<T>
    callback );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public void counter.GetAndDecrement( AsyncCallback<T> callback );
```

♦ **Synchronousの呼出関数:**

```
// Backendless.Counters approach
public int Backendless.Counters.GetAndDecrement( String counterName );
public T Backendless.Counters.GetAndDecrement<T>( String counterName );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public T counter.GetAndDecrement();
```

♦ **Parameter:**

counterName - 減少するcounter 名を指定します。

T - counter 値の数字形式を指定します。byte、 short、 int、 float、 long and doubleを指定することができます。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<int> callback = new AsyncCallback<int>(
  result =>
  {
    System.Console.WriteLine( "[ASYNC] previous counter value is - " + result );
  },
  fault =>
  {
     System.Console.WriteLine( "Error - " + fault );
  } );

Backendless.Counters.GetAndDecrement( "my counter", callback );

IAtomic<int> myCounter = Backendless.Counters.Of<int>( "my counter" );
int counter = myCounter.GetAndDecrement();
System.Console.WriteLine( "[SYNC] previous counter value through IAtomic is - " + counter );

counter = Backendless.Counters.GetAndDecrement( "my counter" );
System.Console.WriteLine( "[SYNC] previous counter value is - " + counter );

long longCounter = Backendless.Counters.GetAndDecrement<long>( "my counter" );
System.Console.WriteLine( "[SYNC] previous counter value is - " + longCounter );
```

8.4. Decrement by 1, return current

♦ **Asynchronousの呼出関数:**

```
// Backendless.Counters approach
public void Backendless.Counters.DecrementAndGet<T>( String counterName, AsyncCallback<T>
    callback );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public void counter.DecrementAndGet( AsyncCallback<T> callback );
```

♦ **Synchronousの呼出関数:**

```
// Backendless.Counters approach
public int Backendless.Counters.DecrementAndGet( String counterName );
public T Backendless.Counters.DecrementAndGet<T>( String counterName );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public T counter.DecrementAndGet();
```

♦ **Parameter:**

counterName – 減少させるするcounter 名を指定します。

T - counter 値の数字形式を指定します。byte、 short、 int、 float、 long and doubleを指定することができます。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<int> callback = new AsyncCallback<int>(
  result =>
  {
    System.Console.WriteLine( "[ASYNC] current counter value is - " + result );
  },
  fault =>
  {
     System.Console.WriteLine( "Error - " + fault );
  } );

Backendless.Counters.DecrementAndGet( "my counter", callback );

IAtomic<int> myCounter = Backendless.Counters.Of<int>( "my counter" );
int counter = myCounter.DecrementAndGet();
System.Console.WriteLine( "[SYNC] current counter value through IAtomic is - " + counter );

counter = Backendless.Counters.DecrementAndGet( "my counter" );
System.Console.WriteLine( "[SYNC] current counter value is - " + counter );

long longCounter = Backendless.Counters.DecrementAndGet<long>( "my counter" );
System.Console.WriteLine( "[SYNC] current counter value is - " + longCounter );
```

8.5. Increment by N, return current

- **Asynchronousの呼出関数:**

```
// Backendless.Counters approach
public void Backendless.Counters.AddAndGet<T>( String counterName, Int64 value,
    AsyncCallback<T> callback );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public void counter.AddAndGet( Int64 value, AsyncCallback<T> callback );
```

- **Synchronousの呼出関数:**

```
// Backendless.Counters approach
public int Backendless.Counters.AddAndGet( String counterName, Int64 value );
public T Backendless.Counters.AddAndGet<T>( String counterName, Int64 value );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public T counter.AddAndGet( Int64 value );
```

- **Parameter:**

counterName - 増加する counter 名を指定します。

value -増加する値を指定します。

T - counter 値の数字形式を指定します。byte、 short、 int、 float、 long and doubleを指定することができます。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

- **呼出関数:**

```
AsyncCallback<int> callback = new AsyncCallback<int>(
  result =>
  {
    System.Console.WriteLine( "[ASYNC] current counter value is - " + result );
  },
  fault =>
  {
     System.Console.WriteLine( "Error - " + fault );
  } );

Backendless.Counters.AddAndGet( "my counter", 1000, callback );

IAtomic<int> myCounter = Backendless.Counters.Of<int>( "my counter" );
int counter = myCounter.AddAndGet( 1000 );
System.Console.WriteLine( "[SYNC] current counter value through IAtomic is - " + counter );

counter = Backendless.Counters.AddAndGet( "my counter", 1000 );
System.Console.WriteLine( "[SYNC] current counter value is - " + counter );

long longCounter = Backendless.Counters.AddAndGet<long>( "my counter", 1000 );
System.Console.WriteLine( "[SYNC] current counter value is - " + longCounter );
```

8.6. Increment by N, return previous

♦ **Asynchronousの呼出関数:**

```
// Backendless.Counters approach
public void Backendless.Counters.GetAndAdd<T>( String counterName, Int64 value,
    AsyncCallback<T> callback );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public void counter.GetAndAdd( Int64 value, AsyncCallback<T> callback );
```

♦ **Synchronousの呼出関数:**

```
// Backendless.Counters approach
public int Backendless.Counters.GetAndAdd( String counterName, Int64 value );
public T Backendless.Counters.GetAndAdd<T>( String counterName, Int64 value );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public T counter.GetAndAdd( Int64 value );
```

♦ **Parameter:**

counterName - 増加する counter 名を指定します。

value -増加する値を指定します。

T - counter 値の数字形式を指定します。byte、 short、 int、 float、 long and doubleを指定することができます。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<int> callback = new AsyncCallback<int>(
  result =>
  {
    System.Console.WriteLine( "[ASYNC] previous counter value is - " + result );
  },
  fault =>
  {
     System.Console.WriteLine( "Error - " + fault );
  } );

Backendless.Counters.GetAndAdd( "my counter", 1000, callback );

IAtomic<int> myCounter = Backendless.Counters.Of<int>( "my counter" );
int counter = myCounter.GetAndAdd( 1000 );
System.Console.WriteLine( "[SYNC] previous counter value through IAtomic is - " + counter );

counter = Backendless.Counters.GetAndAdd( "my counter", 1000 );
System.Console.WriteLine( "[SYNC] previous counter value is - " + counter );

long longCounter = Backendless.Counters.GetAndAdd<long>( "my counter", 1000 );
System.Console.WriteLine( "[SYNC] previous counter value is - " + longCounter );
```

8.7. Conditional update

カウンタの値が指定された値と等しくなったタイミングで、指定された値を代入します。

♦ **Asynchronousの呼出関数:**

```
// Backendless.Counters approach
public void Backendless.Counters.CompareAndSet<T>( String counterName, Int64 expected, Int64
    updated, AsyncCallback<bool> callback );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public void counter.CompareAndSet( Int64 expected, Int64 updated, AsyncCallback<T> callback );
```

♦ **Synchronousの呼出関数:**

```
// Backendless.Counters approach
public bool Backendless.Counters.CompareAndSet( String counterName, Int64 expected, Int64
    updated );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public bool counter.CompareAndSet( Int64 expected, Int64 updated );
```

♦ **Parameter:**

counterName - update対象のcounter 名を指定します。

expected - counterの期待値を指定します。もし現在値が期待値と等しければ "updated" 値に設定されます。

updated - 現在値が期待値と等しければ新しい値を counterに代入します。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<bool> callback = new AsyncCallback<bool>(
  result =>
  {
    System.Console.WriteLine( "[ASYNC] valud has been updated? - " + result );
  },
  fault =>
  {
     System.Console.WriteLine( "Error - " + fault );
  } );

Backendless.Counters.CompareAndSet( "my counter", 1000, 2000, callback );

IAtomic<int> myCounter = Backendless.Counters.Of<int>( "my counter" );
bool updateResult = myCounter.CompareAndSet( 1000, 2000 );
System.Console.WriteLine( "[SYNC] value has been updated? - " + updateResult );

updateResult = Backendless.Counters.CompareAndSet( "my counter", 2000, 3000 );
System.Console.WriteLine( "[SYNC] value has been updated? - " + updateResult );
```

8.8. Get current

カウンタの現在の値を取得します。

♦ **Asynchronousの呼出関数:**

```
// Backendless.Counters approach
public void Backendless.Counters.Get<T>( String counterName, AsyncCallback<T> callback );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public void counter.Get( AsyncCallback<T> callback );
```

♦ **Synchronousの呼出関数:**

```
// Backendless.Counters approach
public int Backendless.Counters.Get( String counterName );
public T Backendless.Counters.Get<T>( String counterName );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public T counter.Get();
```

♦ **Parameter:**

counterName - 現在値を取得する counter 名を指定します。

T - counter 値の数字形式を指定します。byte、 short、 int、 float、 long and doubleを指定することができます。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<int> callback = new AsyncCallback<int>(
  result =>
  {
    System.Console.WriteLine( "[ASYNC] current counter value is - " + result );
  },
  fault =>
  {
     System.Console.WriteLine( "Error - " + fault );
  } );

Backendless.Counters.Get( "my counter", callback );

IAtomic<int> myCounter = Backendless.Counters.Of<int>( "my counter" );
int counter = myCounter.Get();
System.Console.WriteLine( "[SYNC] current counter value through IAtomic is - " + counter );

counter = Backendless.Counters.Get( "my counter" );
System.Console.WriteLine( "[SYNC] current counter value is - " + counter );

long longCounter = Backendless.Counters.Get<long>( "my counter" );
System.Console.WriteLine( "[SYNC] current counter value is - " + longCounter );
```

8.9. Reset

カウンタをリセットします。0が代入されます。

♦ **Asynchronousの呼出関数:**

```
// Backendless.Counters approach
public void Backendless.Counters.Reset( String counterName, AsyncCallback<Object> callback );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public void counter.Reset( AsyncCallback<Object> callback );
```

♦ **Synchronousの呼出関数:**

```
// Backendless.Counters approach
public void Backendless.Counters.Reset( String counterName );

// IAtomic approach
IAtomic<T> counter = Backendless.Counters.Of<T>( String counterName );
public void counter.Reset();
```

♦ **Parameter:**

counterName - resetする counter 名を指定します。

callback - Asynchronous 関数の結果を取得するコールバック関数を指定します。

♦ **呼出関数:**

```
AsyncCallback<Object> callback = new AsyncCallback<Object>(
  result =>
  {
    System.Console.WriteLine( "[ASYNC]counter has been reset" );
  },
  fault =>
  {
     System.Console.WriteLine( "Error - " + fault );
  } );

Backendless.Counters.Reset( "my counter", callback );

IAtomic<int> myCounter = Backendless.Counters.Of<int>( "my counter" );
myCounter.Reset();
System.Console.WriteLine( "[SYNC] counter has been reset" );

Backendless.Counters.Reset( "my counter" );
System.Console.WriteLine( "[SYNC] counter has been reset" );
```