# coach-admin-api Documentation

***Release 1.0.2***

**Garrard Kitchen**

December 15, 2015

Contents

Contents:

CHAPTER 1

# Getting Started

## 1.1 Conventions used

**Bold** - Mostly hyperlinks or just plain text for highlighting the current API page. Eg. in Tenant page there is no need to make link to itself it is just bold text.

*Italic* - Indicates highlighted words or meanings.

`id` - Indicates the code properties and assigned values like true or null.

**Hint:** Indicates some text that is a note to give more information on some subject.

**Warning:** Indicate some text that warns of something that can happen and will not result in error.

**Danger:** Indicate some text that will cause some errors and developers should be aware of its implication(s).

## 1.2 Coach

The *Coach* has two main parts:

- *Console* the administrative part where the Tenant hierarchy can be created, connection to Recorder(s) and assign different Media Players for playback, and set Scheduled Tasks.
- *QM* or Quality Monitoring part for media file monitoring standards to be established and then used to analytically evaluate files with the inclusion of carefully targeted coaching tools.

**Note:** This *REST API* is fully implemented for administrative part of *Coach* or *Console* part. There are few settings that can be set for *QM* but they are all assigned through *Console API*.

## 1.3 Formats

Currently supported format are *JSON* (JavaScript Object Notation) and *XML*. Requests are valid as long as they are sent as HTTP Header Content-Type *application/json* or *application/xml*, and result response format is same as sent request format.

## 1.4 Authentication

With installation of *Coach* each customer will be provided with unique *API Key* and *API Secret*. The API *Key* and *Secret* are need for authentication to the *Coach REST API* and they are per Tenant base, so for each call to *REST API* service is used same combination of *Key* and *Secret*.

**Warning:** It is up to developers using *Console REST API* to restrict or allow user to be able to gain access to *REST API*. From *REST API* point of view there is no distinction what user is using *REST API* since the same *Key* and *Secret* is used to gain access to the *Console API*.

### 1.4.1 A RESTful example

The *HTTP Header* is where you include the `key` and `secret` *fields*. Please note the names of these *fields*; `consumer_key` and `consumer_secret`:

```
consumer_key: neFh2vtr2sKH1tvsp0O6
consumer_secret: 86h1qQEAhwlXydNJTLQj9KK3e5DUZUhoYjLsJKv72k44ZkmF2h
```

When issuing `GET http://mydomain.com/aspire/api/v1/1001/users HTTP/1.1` with the above *HTTP Header* the result will be *JSON* describing all the *Users* belonging to *Tenant* 1001

**Hint:** The `key` and `secret` for the *Host Tenant* are created during the installation. You can subsequently access and regenerate these values from within *Coach*. Please refer to the *Coach Installation* documentation for further information.

## 1.5 Create, Update and Lookups

The *REST API* is constructed in a way that to create or update some resource it is needed to get resource by id that will populate the resource form with data, so it can be submitted to the server after changes made by user on form.

With get by id you'll get new or persisted instance of entity, with all default values already set (only on create) and also all the *lookup* (read more about about lookups in note below) data will be sent to client.

**Hint:** The lookup in this sense is collection of `key` and `value` pairs needed to set some referenced id in some resource. It is used to populate drop boxes, radio buttons group,... The *key* of selected or checked item should be used to set some referenced value.

Example would be to create *Team*, there is need to set it to a belonging *Unit* (`unitId`). The *Team* lookup `units` will provide you with all *Units* `ids` as `key` and *Unit* `names` as `value` of lookup so it is to set *Team's* belonging *Unit* by choosing one *key* and setting it to *Team's* `unitId`.

The *lookup* enumeration keys are *integers* and *ids* used are *guids*.

To set up the form on adding/creating new entity is needed to use get by id and for `id` is needed to set and `empty guid` or `"00000000-0000-0000-0000-000000000000"`, this will give you new instance of an entity with default values set and lookups for references loaded so that it can be use for form creation.

## 1.6 REST API Methods and Models

The API methods are the way to obtain data from *REST API* and models are way this data is represented.

### 1.6.1 Models

The Models are representation of some resource as its properties and values assigned to it. The *Coach API* uses two kind of *Models*:

#### Domain Models

Used to for create and update domain model, but also for constructing and getting persisted **Domain Model**.

#### List Model

The light and normalized version of **Domain Model** used for lists of resource (usually represented as grid or tabular data) and with fewer properties than **Domain Models** and instead of reference `ids` uses human readable `name` to show in list item.

---

**Hint:** Note that **Domain Models** and **List Models** are described in lot more details about its properties and behaviors on each *API* resource page.

---

### 1.6.2 Methods

The *REST API* methods are analogous to *HTTP Verbs*. For most of resources in *REST API* you will have this five methods to get most common CRUD (Create Read Update Delete) data from and to server.

#### GET

Returns a list of all items for requested resource. It is sent as **List Model** and it is fully read-only data. If there is error on server, it will send error response instead.

#### GET /:id

Returns a **Domain Model** representation of resource for particular `id`. Embedded with resources are *lookups*, and if is used `empty guid` it will already set all default values. If there is error on server, it will send error response instead.

#### POST

Sends the newly created resource **Domain Model** and after saving it to database returns all the created values (now persisted) as it was called by `GET :id`. If there is error on server, it will send error response instead.

#### PUT

Sends the updated resource **Domain Model** and after saving it to database returns all the updated values (now persisted) as it was called by `GET :id`. If there is error on server, it will send error response instead.

#### DELETE

Sends the resources `id` to server to delete resource. After deletion if everything went well it will not send anything, but if there is error on server, error will be sent as response.

**Warning:** Note that all resources don't have support for all methods described before.
- The *Tenant* is currently supports only `PUT` or update (this will change with implementation of multi-tenancy).
- The *Tenant Tree* is supports only `GET` method that or will bring only a Tenant Tree Items.
- The *License* is now supports only `GET` method or will only obtain license from Licensing Server.

## 1.7 HTTP PUT and DELETE Issue

There are known issues with *HTTP* verbs `PUT` and `DELETE`. The `PUT` and `DELETE` were not supported on older browsers and `PUT` and `DELETE` can be disabled or not disabled by default in *IIS Web Server*.

### 1.7.1 Activate PUT and DELETE on IIS Web Server

To activate *HTTP* `PUT` and `DELETE` on *IIS Web Server* you need to add this part of configuration to the `web.config` of web application where *Coach REST API* is hosted.

**Hint:** *Coach REST API* is hosted in same location as *Coach* Silverlight application is hosted.

For *IIS 7* and above you can use this configuration to enable `PUT` and `DELETE`:

```
<system.webServer>
 <validation validateIntegratedModeConfiguration="false" />
 <modules runAllManagedModulesForAllRequests="true" />
 <handlers>
   <remove name="ExtensionlessUrlHandler-Integrated-4.0" />
   <add name="ExtensionlessUrlHandler-Integrated-4.0"
   path="*."
   verb="GET,HEAD,POST,DEBUG,PUT,DELETE"
   modules="IsapiModule"
   scriptProcessor="C:\Windows\Microsoft.NET\Framework64\v4.0.30319\aspnet_isapi.dll"
   resourceType="Unspecified"
   requireAccess="Script"
   preCondition="classicMode,runtimeVersionv4.0,bitness64"
   responseBufferLimit="0" />
 </handlers>
</system.webServer>
```

**Warning:** Please be aware that this piece of configuration is as-is, it is just example of enabling PUT and DELETE. For more information on enabling the PUT and DELETE in *IIS* please use *IIS* documentation or in house System Administrator if you have.

### 1.7.2 Using API with disabled PUT and DELETE

Activating the *HTTP* `PUT` and `DELETE` is optional step, it is recommended, but optional. If from some case it is problem or it is in house rule to not allow `PUT` and `DELETE` or you still supporting browser that are not supporting this verbs then there is a way to sent *HTTP* `PUT` and `DELETE` as `POST` but with a *HTTP Header* directive `X-HTTP-Method-Override`.

### 1.7.3 UPDATE via POST

To use `UPDATE` via `POST` you need to send `POST` request and in *HTTP Header* set:

```
X-HTTP-Method-Override: UPDATE
```

### 1.7.4 DELETE via POST

To use `DELETE` via `POST` you need to send `POST` request and in *HTTP Header* set:

```
X-HTTP-Method-Override: DELETE
```

## 1.8 TODO List

- Finish Multi-Tenancy for Tenant via Create/Update/Delete + update documentation.
- Create Tenant Tree of only Unit/Team/Agent items for Schedule Level.

CHAPTER 2

# C# Wrapper

**Danger:** Please make sure that you have read *Getting Started* because even the implementation is very different then default default *REST* approach, there are some concepts and issues described in much more details than it is here and some are assumed so there are no details at all.

The *C# Wrapper Library* that hides and abstracts the REST details under cover of wrapper and allows developers to use *REST API* in native way or in this case C# way. The *C# Wrapper Library* behind scene uses *JSON* as content media type.

The *C# Wrapper Library* also abstracts the authentication, *HTTP* `PUT` and `DELETE` verb issue in more easier way.

## 2.1 Setup and Dependencies

### 2.1.1 Setup

To setup *C# Wrapper Library* there is a need to add the reference to `Qualtrak.Coach.API.Wrappers.CSharp.dll` provided with installation of *Coach*. Note that project framework is needed to be *.NET 4.5*, but note that *.NET 4.5 Client Profile* is not supported.

### 2.1.2 Dependencies

The dependencis to run the *C# Wrapper Library* are:

- [JSON.NET](https://json.codeplex.com/releases) (actually the compiled name is `Newtonsoft.Json.dll`).
- **CHECK OTHER DEPENDENCIES**

**Warning:** The *C# Wrapper Library* uses Json.NET 4.5 Release 7 but any newer version should work properly.

## 2.2 Authentication

Unlike send it in *HTTP Header* like in normal *REST* way this is abstract, but information is needed so it is a part of each C# Wrapper method. There is a need for providing the `key` and `secret` for each call to the *REST API*.

## 2.3 Response Content and API Wrapper

Response Content is the data that is result of some request, and API wrapper is the way to send requests to *REST API*.

### 2.3.1 Response Content

It is class `ResponseContent` that holds the *REST API* response message as its content.

### 2.3.2 Non-Generic Version

The non-generic version of `ResponseContet` has two properties:

#### Error

The `Error` property and it is an error response when some known error occurred due validation or some unknown exception occurred on server. The `Error` property is actual Domain Model, read more in Client Errors

#### HttpStatusCode

The `integer` representation of *HTTP Staus Code* (eg. 200).

### 2.3.3 The Generic Version

The generic version `ResponseContent<T>` inherits from `ResponseContent` so it has its properties and also:

#### Result<T>

The `Result<T>` is a actual result data sent as response content, and the generic `T` can be List or Domain Model.

### 2.3.4 API Wrapper

The *API Wrapper* is actually interface `ITreeApiWrapper<TModel, TList>` used is mostly all CRUD resources for *Coach Console API*.

The `ITreeApiWrapper<TModel, TList>` has this methods:

#### ResponseContent<ICollection<TList>> GetAll()

Method used for getting list of all request items that are collection of entity List Model. The `Result<T>` will be sent if get all was successful if there is an error then the `Error` property will hold the error message.

#### ResponseContent<TModel> GetById(Guid id)

Method used for getting a new or persisted instance of entity Domain Model. Use this method for getting default values on add and lookup data for both add and edit of entity. The `Result<T>` will be sent if get by id was successful if there is an error then the `Error` property will hold the error message.

#### ResponseContent<TModel> Create(TModel entity)

Method used for creating the resource. It expects created Domain Model to be sent and it returns this persisted Domain Model as it is called by `GetById`. The `Result<T>` will be sent if create was successful if there is an error then the `Error` property will hold the error message.

#### ResponseContent<TModel> Update(TModel entity, bool updateViaPost = false)

Method used for updating already persisted the resource. It expects updated Domain Model to be sent and it returns this updated Domain Model as it is called by `GetById`. The `Result<T>` will be sent if update was successful if there is an error then the `Error` property will hold the error message.

---

**Hint:** The parameter `updateViaPost` is set by default to false and it will be using `PUT` on request, if this is issue, you can set it to `true` and will be using `PUT` via `POST`.

---

#### ResponseContent Delete(Guid id, bool deleteViaPost = false)

Method used to delete resource by sending `id`. It returns non-generic `ResponseContent` so there is no result just `Error`. So if `Error` is not `null` then something went wrong on server.

---

**Hint:** The parameter `deleteViaPost` is set by default to false and it will be using `DELETE` on request, if this is issue, you can set it to `true` and will be using `DELETE` via `POST`.

---

**Warning:** Note that all resources doesn't implement `ITreeApiWrapper<TModel, TList>` like *Tenant*, *Tenant Tree* and *License*.

CHAPTER 3

# Client Errors

The description of client side errors that can get from *REST API* server on known logic errors and unknown exceptions.

## 3.1 Error Model

Represents the **Error** model as value object with available properties.

| Name | Description | Type |
|---|---|---|
| `id` | Representing **Error** identifier from API Logs. | `guid` |
| `message` | The **Error** message it can be descriptive for known error and generic for unknown exceptions. | `string` |
| `description` | The **Error** description mainly a generic text for unknown exception. | `string` |
| `httpStatusCode` | The HTTP Status Code, eg. 200. | `integer` |
| `requestArguments` | The collection of sent Request Arguments. | `array(RequestArgument)` |

**Hint:** The **Error** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Error** properties are capitalized (eg. `Id`, `Name`,..)!

Note that C# Wrapper has for **Error** Model overridden a `ToString()` method and calling instance of error like `error.ToString()` will result with displaying the **Error** and all **Requested Arguments**

**Warning:** Note that only simple **Request Arguments** will be sent like `integer`, `string`, `boolean`,... The complex types are currently not supported to be sent as **Error Request Arguments**.

## 3.2 Request Argument Model

Represents the **Request Argument** model as value object with available properties.

| Name | Description | Type |
|---|---|---|
| `name` | The **Request Argument** name, eg. `id`. | `string` |
| `value` | The **Request Argument** value for `name` **Request Argument**. | `string` |

**Hint:** The **Error** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Error** properties are capitalized (eg. `Id`, `Name`,..)!

## 3.3 Known Errors

The list of **Known Errors** that can happen while interacting with *Coach REST API*.

**Hint:** Note that this errors are generic description of server side known errors and errors sent to client will be with proper and very descriptive error messages, due to some logic and according to what arguments are sent when error occurred. Many errors are more described in particular entity's documentation, mostly around its Domain Model.

| Known Error | Description |
|---|---|
| **Missing Consumer Key and/or Secret** | Consumer key and/or secret is not sent with HTTP Header. |
| **Unauthorized Key and/or Secret** | Consumer key/secret combination is not known and hence unauthorized. |
| **Invalid Tenant Code** | The tenant code provided through URL is not valid. Must be 1000 and greater! |
| **Unknown Tenant Code** | The tenant code is not existing. |
| **Unknown Content-Type** | The Request Header must include a valid and supported Content-Type media format (JSON or XML). |
| **Exceeded License Total For Agents** | The licenses exceeded the total of purchased licenses. Note one active user as agent is one license! |
| **Unknown Reference Entity Id** | The sent referenced entity GUID Id is unknown or empty GUID. Reference to entity can be established. |
| **Unique Username** | Validates the uniqueness of User's username. There cannot be two same usernames for tenant. |
| **Required Field** | Validates that required field or property has proper value assigned to it. |
| **Invalid Id** | The Id of entity is empty GUID. |
| **Invalid Date Range** | The date range of start and end date is invalid when end date is greater than start date. |
| **Incorrect Period Type** | The incorrect combination usage of schedule period type and occurrence. |
| **Incorrect Numeric** | The applied numeric value is not correct whether of crossing some range, negative number, greater than 0,... |
| **Incorrect Date** | The applied data is no correct due some date logic. |
| **Delete While Having Reference** | The entities that are referenced by some other entities are not allowed to be deleted. |

CHAPTER 4

# Release History

## 4.1 1.0.2 (2015-11-20)

- Initial release as an OSS project with **Coach 6.0**

## 4.2 1.0.1 (2015-01-01)

- Several minor bug fixes

## 4.3 1.0.0 (2014-01-01)

- Initial release with **Coach 5.0**

CHAPTER 5

# Roles

There are four built-in **Roles** contained in **Coach**. These are:

- System Administrator
- Application Administrator
- Manager
- Agent

**Danger:**
Currently **Roles** can only be assigned through *User*, but only *administrative* **Roles** (*System Administrator*, *Application Administrator*).
The **Roles** *Manager* and *Agent* are not assigned as **Role** but rather as *Unit*/*Team managership* for *Manager* and/or *Team membership* for *Agent*.
The *User* is *Manager* as long as he or she is assigned as manager of at least one or more *Unit* or *Team*.
The *User* is *Agent* as long as he or she is assigned as member of at least one or more *Team*.

Each Role has a different purpose with correspondingly different levels of access to **Coach** Console and **Coach** QM. These levels of access are called *Permissions*. Multiple **Roles** can be assigned to the same User, the differing Permissions for differing **Role** being combined with one another

## 5.1 System Administrator

During the **Coach** installation process a *User* with the **Role** of *System Administrator* is created. This *System Administrator* can only enter **Coach** *Console* for *Tenant* 1000, the original host *Tenant*. Here they have full permissions within **Coach** *Console* for every extant *Tenant* and can add new *Tenants* as well as view, edit and delete any that have been created.

Apart from the *System Administrator* created during **Coach**'s installation *Users* can only be added manually to the *System Administrator* **Role**. The **Role** can be assigned to any *User* but it can only be assigned by, and is only visible to, *Users* who already have the **Role**. Due to the importance of this **Role** for the **Coach** application, a *System Administrator* cannot un-assign themself from the **Role** of *System Administrator* (or any other **Role** they may have been given) – only a fellow *System Administrator* can perform this action, thereby ensuring that at least one *User* always has the *System Administrator* **Role** in each installation. For the same reason a *System Administrator* can only be deactivated by another *System Administrator*.

## 5.2 Application Administrator

When a new *Tenant* is created an *Application Administrator* must be created. An *Application Administrator* can access both **Coach** Console and **Coach** *QM* for the *Tenant* they were created for. In **Coach** Console they have full permissions excepting the ability to add new *Tenants* or delete their own. In **Coach** *QM* they have full permissions except that they cannot create Evaluations or **Coaching Sessions**.

Apart from the *Application Administrator* created with a new *Tenant* further *Users* can only be added manually to the *Application Administrator* **Role**. Due to the importance of this **Role** for a *Tenant*, an *Application Administrator* cannot un-assign themselves from the **Role** of *Application Administrator* (or any other **Role** they may have been given) – only a fellow *Application Administrator* (or a *System Administrator*) can perform this action, thereby ensuring that at least one *User* always has the *Application Administrator* (or *System Administrator*) **Role** within each *Tenant*. For the same reason an *Application Administrator* can only be deactivated by another *Application Administrator* (or a *System Administrator*).

## 5.3 Manager

A *Manager* can only enter **Coach** *QM*. Here they can view everything but can only create, edit and delete *Evaluations*, *Coaching Sessions* and *Reports* for the *Users* they manage. *Users* are automatically assigned to the *Manager* **Role** when they are made *Manager* of a *Team* or Unit.

## 5.4 Agent

An *Agent* can only enter **Coach** *QM* and can only access *Evaluations* and *Coaching Sessions* created for them. *Users* are automatically assigned to the *Agent* **Role** when they are joined to a *Team*.

CHAPTER 6

# Tenant

The **Tenant** is the organization. A **Tenant** must be created before any other step can be taken and this step will be performed during the initial installation process. The structure of the company can be replicated within the make-up of the **Tenant** through *Unit* and *Teams*.

Whenever a **Tenant** is created, part of the process involves the creation of a unique administrative *User*. For the original host **Tenant**, setup during the **Coach** installation process, this *User* is a *System Administrator*. With all additional **Tenants** this built in *User* will be an *Application Administrator*. See more information in [Roles](/v1/roles).

**Danger:** Only a System Administrator has the permission to create, edit and delete **Tenants**, though an *Application Administrator* can edit all of their own **Tenant's** properties.

**Danger:**
Note that multi-tenancy or create and delete of **Tenant** is currently not supported through *API*, but it will be soon!
With multi-tenancy implementation except *API* breaking changes for *REST* and *C# Wrapper* approach!!!

## 6.1 Tenant Domain Model

Represent the **Tenant** domain model with available properties and its behaviors.

**Note:** Note that domain model is used for write methods *POST* (*Create*) and *PUT* (*Update*) and as result of read-only method *GET/:id* (*GetById*).

| Name | Description | Type | Required | Read-only | Default |
|---|---|---|---|---|---|
| `id` | Representing **Tenant** identifier. | `guid` | yes | yes | |
| **Tenant Details** | | | | | |
| `name` | The name of **Tenant**. | `string(50)` | yes | no | |
| `description` | The **Tenant** description. | `string(500)` | no | no | |
| `email` | The **Tenant** email. | `string(500)` | no | no | |
| `phone` | The **Tenant** phone number. | `string(500)` | no | no | |
| `address` | The **Tenant** residing address. | `string(500)` | no | no | |
| `mainContact` | The **Tenant** main contact person. | `string(500)` | no | no | |
| `isActive` | Denotes whether the **Tenant** state is active or inactive. | `boolean` | yes | no | active (`true`) |
| `isDeleted` | Denotes whether the **Tenant** state is deleted or not. | `boolean` | yes | yes | not deleted (`false`) |
| **Branding** | | | | | |
| `customNameForTenant` | Branded replacement name for word "Tenant", affects *QM* UI. | `string(300)` | no | no | `Tenant` |
| `customNameForUnit` | Branded replacement name for word "Unit", affects *QM* UI. | `string(300)` | no | no | `Unit` |
| `customNameForTemplate` | Branded replacement name for "Template", affects *QM* UI. | `string(300)` | no | no | `Template` |
| **Licensing** | | | | | |
| `customerId` | | `string(500)` | no | no | |
| `customerCode` | | `string(500)` | no | no | |
| `licenseExpiryDate` | The **Tenant** license expiry date. | `datetime` | N/A | yes | *null* |
| `totalLicenses` | The number of *User* total licenses for **Tenant**. | `integer` | N/A | yes | |
| `usedLicenses` | The number of used licenses for **Tenant**. One *User* equals one license. | *integer* | N/A | yes | |
| **API** | | | | | |
| `apiKey` | The API key for **Tenant**. | `string` | no | no | |
| `apiSecret` | The API secret for **Tenant**. | `string` | no | no | |

**Note:** The **Tenant** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **User** properties are capitalized (eg. `Id`, `Name`,..)!

## 6.2 Get Tenant by Id

The **Tenant** by requested Id.

### 6.2.1 Default REST approach

GET /api/v1/:tenantCode/tenants/:tenantCode

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in [Authentication](/v1/authentication).

#### Return value

- If there is no error: `JSON` as the [Tenant Domain Model](/v1/tenant#tenant-model) object.
- If there is an error: `JSON` as the [error](/v1/client-errors#error-model) object.

### 6.2.2 C# Wrapper approach

```
TenantWrapper(int tenantCode, string apiKey, string apiSecret).GetByCode(int tenantCode);
```

#### Parameters

- `tenantCode` The **Tenant** code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.

#### Return value

- If there is no error: `ResaultContent<Tenant>.Result` object as the [Tenant Domain Model](/v1/tenant#tenant-model).
- If there is an error: `ResaultContent<Tenant>.Error` object. See more in [Client Errors](/v1/client-errors).

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

TenantWrapper tenantWrapper = new TenantWrapper(tenantCode, key, secret);
ResponseContent<Tenant> response = tenantWrapper.GetByCode(tenantCode);

if (response.Result != null)
{
     // Use Result as requested Tenant for displaying.
     Tenant tenant = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 6.3 Update Tenant

Updates already existent **Tenant**.

### 6.3.1 Default REST approach

PUT /api/v1/:tenantCode/tenants/:tenantCode

#### Parameters

- `tenantCode` The **Tenant** code, a valid `integer` greater or equal to 1000.
- `tenant` `JSON` representation of [Tenant Domain Model](/v1/tenant#tenant-model) sent via *Request HTTP Header*.

**Danger:**
Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*.
See more in [Authentication](/v1/authentication).
If you don't want to have in Web Server turned on the *PUT* verb method read more in *Getting Started*.

#### Return value

- If there is no error: `JSON` representation of uodated **Tenant** as the [Tenant Domain Model](/v1/tenant#tenant-model).
- If there is an error: `JSON` [error](/v1/client-errors#error-model) object.

### 6.3.2 C# Wrapper approach

```
TenantWrapper(int tenantCode, string apiKey, string apiSecret).Update(Tenant tenant, bool updateViaPo
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `tenant` The **Tenant** model constructed from **Tenant properties** and `Id` must be provided in it. If not `ArgumentException` will be thrown!
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `PUT` method. Default is `false` or use `PUT` method!

#### Return value

- If there is no error: `ResaultContent<Tenant>.Result` object as the [Tenant Domain Model](/v1/tenant#tenant-model).

- If there is an error: `ResaultContent<Tenant>.Error` object. See more in [Client Errors](/v1/client-errors).

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

TenantWrapper tenantWrapper = new TenantWrapper(tenantCode, key, secret);
Tenant tenant = tenantWrapper.GetByCode(tenantCode).Result;
tenant.name = "Tenant Updated";

// Update via PUT method (default).
ResponseContent<Tenant> response = tenantWrapper.Update(tenant);

// Update via POST method (use true argument).
// ResponseContent&lt;Tenant&gt; response = tenantWrapper.Update(tenant, true);

if (response.Result != null)
{
     // Use Result of updated Tenant for display.
     Tenant updatedTenant = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

CHAPTER 7

# Unit

The **Unit** is part of *Tenant*. A **Unit** may be a department or contact center within an organization. Each **Unit** may contain one or more other organizational sub-units and act as a **"Parent Unit"** to them.

## 7.1 Unit Domain Model

Represent the **Unit** domain model with available properties and its behaviors.

**Note:** Note that domain model is used for write methods *POST* (*Create*) and *PUT* (*Update*) and as result of read-only method *GET/:id* (*GetById*).

| Name | Description | Type | Required | Read-only | Default |
|---|---|---|---|---|---|
| `id` | Representing **Unit** identifier. | `guid` | yes | yes | |
| `name` | The name of **Unit**. | `string(50)` | yes | no | |
| `description` | The description of **Unit**. | `string(255)` | no | no | |
| `isActive` | Denotes whether the **Unit** state is active or inactive. | `boolean` | yes | no | active (`true`) |
| `isDeleted` | Denotes whether the **Unit** state is deleted or not. | `boolean` | yes | [partially] | not deleted (`false`) |
| `parentUnitId` | Represent the hierarchical parent of **Unit**. If there is no parent **Unit**, then **Unit** is root. | `guid` | no | no | Unit is root (`null`) |
| `managers` | The **Unit** assigned [Managers (User)](/v1/user). | `array(guid)` | no | no | |
| **Lookups** | | | | | |
| `unitsLookup` | The dictionary of active and not deleted **Units**, needed for choosing parent **Unit** and setting `parentUnitId`. | `dictionary(guid, string)` | N/A | N/A | N/A |
| `usersLookup` | The dictionary of active and not deleted *Users* needed for setting the **Unit** `managers`. | `dictionary(guid, string)` | N/A | N/A | N/A |

**Note:** The **Unit** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Unit** properties are capitalized (eg. `Id`, `Name`,..)!

**Warning:**
**Active and Deleted Logic**

When **Unit** is deleted by [Delete command](/v1/unit#unit-delete) and it is flagged as `isDeleted` as `true` and also by default it is flagged `isActive` as `false`. Note that status `isActive` will remain "*locked*" until the **Unit's** `isDeleted` state is updated to `false` or *not deleted* anymore. Then `isActive` is "*unlocked*" and can be changed. If the **Unit** is *deleted* and on update is tried to change `isActive` property, server will silently ignore sent `isActive` property.

[Unit Manager(s)](/v1/user) and parent **Unit** can be set to only *active* and *not deleted* **Units**. If is sent otherwise to *inactive* and/or *deleted* **Unit**, server will silently ignore those assignments.

## 7.2 Unit List Model

Represent the **Unit** list model with available properties.

**Note:**

The list model used only to list **Units** with *GET* (*GetAll*) method.
Note that list model can change by adding/removing properties depending what users of *Coach REST API* will need in future.

| Name | Description | Type |
|---|---|---|
| `id` | Representing **Unit** identifier. | `guid` |
| `name` | The name of **Unit**. | `string` |
| `description` | The description of **Unit**. | `string` |
| `isActive` | Denotes whether the **Unit** state is active or inactive. | `boolean` |
| `isDeleted` | Denotes whether the **Unit** state is deleted or not. | `boolean` |
| `parentUnitName` | The name of **Unit** parent. Needed for representing the parent unit in list of Units. | `string` |

**Note:** The **Unit** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Unit** properties are capitalized (eg. `Id`, `Name`,..)!

## 7.3 List of Units

The list of **Units** for current *Tenant*.

### 7.3.1 Default REST approach

GET /api/v1/:tenantCode/units

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.

**Danger:** Remember to add *API Key* as *customer key* and *API Secret* as *customer secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` array of [Unit List Model](/v1/unit#unit-list-model).
- If there is an error: `JSON` *Client Errors* object.

### 7.3.2 C# Wrapper approach

```
UnitWrapper(int tenantCode, string apiKey, string apiSecret).GetAll();
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.

#### Return value

- If there is no error: `ResaultContent<ICollection<UnitList>>.Result` object as collection of the [Unit List Model](/v1/unit#unit-list-model).
- If there is an error: `ResaultContent<ICollection<UnitList>>.Error` object. See more in *Client Errors* .

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<Unit, UnitList>unitWrapper = new UnitWrapper(tenantCode, key, secret);
ResponseContent<ICollection<UnitList>> response = unitWrapper.GetAll();

if (response.Result != null)
{
     // Use Result as List of Units for displaying.
     ICollection&lt;UnitList&gt; units = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 7.4 Get Unit by Id

The **Unit** by requested Id for current *Tenant*.

### 7.4.1 Default REST approach

GET /api/v1/:tenantCode/units/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Unit** id, a valid and non-empty `guid`.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` as the [Unit Domain Model](/v1/unit#unit-model) object.
- If there is an error: `JSON` as the *Client Errors* object.

### 7.4.2 C# Wrapper approach

```
UnitWrapper(int tenantCode, string apiKey, string apiSecret).GetById(Guid id);
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `id` The **Unit** id, a valid and non-empty `guid`.

#### Return value

- If there is no error: `ResaultContent<Unit>.Result` object as the [Unit Domain Model](/v1/unit#unit-model).
- If there is an error: `ResaultContent<Unit>.Error` object. See more in *Client Errors* .

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid unitId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<Unit, UnitList> unitWrapper = new UnitWrapper(tenantCode, key, secret);
ResponseContent<Unit> response = unitWrapper.GetById(unitId);

if (response.Result != null)
{
     // Use Result as requested Unit for displaying.
     Unit unit = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 7.5 Create Unit

The creation of new **Unit** for current *Tenant*.

### 7.5.1 Default REST approach

POST /api/v1/:tenantCode/units

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `unit` `JSON` representation of [Unit Domain Model](/v1/unit#unit-model) sent via *Request HTTP Header*.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` representation of newly created **Unit** as the [Unit Domain Model](/v1/unit#unit-model).
- If there is an error: `JSON` *Client Errors* object.

### 7.5.2 C# Wrapper approach

```
UnitWrapper(int tenantCode, string apiKey, string apiSecret).Create(Unit unit);
```

### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `unit` The **Unit** model constructed from [Unit Domain Model](/v1/unit#unit-model).

### Return value

- If there is no error: `ResaultContent<Unit>.Result` object as the [Unit Domain Model](/v1/unit#unit-model).
- If there is an error: `ResaultContent<Unit>.Error` object. See more in *Client Errors* .

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<Unit, UnitList> unitWrapper = new UnitWrapper(tenantCode, key, secret);
// Get default data and lookup for units
Unit newUnit = unitWrapper.GetById(new Guid()).Result;
newUnit.Name = "Unit created from test";
newUnit.Description = "Unit created from test description.";
// Set parent Unit key from units lookup key.
newUnit.ParentUnitId = newUnit.UnitsLookup.FirstOrDefault().Key;
ResponseContent<Unit> response = unitWrapper.Create(newUnit);

if (response.Result != null)
{
     // Use Result as newly created Unit for display.
     Unit unit = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 7.6 Update Unit

Updates already existent **Unit** for current *Tenant*.

### 7.6.1 Default REST approach

PUT /api/v1/:tenantCode/units/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Unit** id, a valid and non-empty `guid`.
- `unit` `JSON` representation of [Unit Domain Model](/v1/unit#unit-model) sent via *Request HTTP Header*.

**Danger:**
Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.
If you don't want to have in Web Server turned on the `PUT` verb method read more in *Getting Started*.

#### Return value

- If there is no error: `JSON` representation of uodated **Unit** as the [Unit Domain Model](/v1/unit#unit-model).
- If there is an error: `JSON` *Client Errors* object.

### 7.6.2 C# Wrapper approach

```
UnitWrapper(int tenantCode, string apiKey, string apiSecret).Update(Unit unit, bool updateViaPost = f
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `unit` The **Unit** model constructed from [Unit Domain Model](/v1/unit#unit-model) and `Id` must be provided in it. If not `ArgumentException` will be thrown!
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `PUT` method. Default is `false` or use `PUT` method!

#### Return value

- If there is no error: `ResaultContent<Unit>.Result` object as the [Unit Domain Model](/v1/unit#unit-model).
- If there is an error: `ResaultContent<Unit>.Error` object. See more in *Client Errors* .

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid unitId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<Unit, UnitList> unitWrapper = new UnitWrapper(tenantCode, key, secret);
Unit unit = unitWrapper.GetById(unitId).Result;
```

```
unit.Name = "Unit updated from test";
unit.Description = "Unit updated from test description.";
// Set parent Unit key from units lookup key.
unit.ParentUnitId = unit.UnitsLookup.FirstOrDefault().Key;

// Update via PUT method (default).
ResponseContent<Unit> response = unitWrapper.Update(unit);

// Update via POST method (use true argument).
// ResponseContent&lt;Unit&gt; response = unitWrapper.Update(unit, true);

if (response.Result != null)
{
     // Use Result of updated Unit for display.
     Unit updatedUnit = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 7.7 Delete Unit

Deletes existent **Unit** for current *Tenant*.

> **Warning:**
> Note that if **Unit** is parent to the other **Units** or there are any *Teams* belonging to it or assigned *Unit Managers* then **Unit** will not be deleted but flagged as `isDeleted`. When **Unit** is deleted it can be undeleted by setting `isDeleted` to `false` while updating **Unit**. \
> If **Unit** has no child **Units**, *Teams* or *Unit Managers*, it will be deleted permanently.

### 7.7.1 Default REST approach

DELETE /api/v1/:tenantCode/units/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Unit** id, a valid and non-empty `guid`.

> **Danger:**
> Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.
> If you don't want to have in Web Server turned on the `DELETE` verb method read more in *Getting Started*.

#### Return value

- There is no return value except if there is an error, the `JSON` *Client Errors* object.

### 7.7.2 C# Wrapper approach

```
UnitWrapper(int tenantCode, string apiKey, string apiSecret).Delete(Guid id, bool updateViaPost = fa
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `id` The **Unit** id, a valid and non-empty `guid`.
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `DELETE` method. Default is `false` or use `DELETE` method!

#### Return value

- If there is no error: no return value or `void`.
- If there is an error: `ResaultContent<Unit>.Error` object. See more in *Client Errors* .

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid unitId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<Unit, UnitList> unitWrapper = new UnitWrapper(tenantCode, key, secret);
// Delete via DELETE method (default).
ResponseContent response = unitWrapper.Delete(unitId);

// Delete via POST method (use true argument).
// ResponseContent response = unitWrapper.Delete(unitId, true);

if (response.Error != null)
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

CHAPTER 8

# Team

A **Team** is the 'end point' to a branch of your organization and it is the only element of an organization that can have *Agents* assigned to it.

**Note:** A **Team** can only be added to a *Unit* or a sub-Unit. \ A **Team** cannot be added to a reversibly deleted and/or inactive *Unit* or sub-Unit.

## 8.1 Team Domain Model

Represent the **Team** domain model with available properties and its behaviors.

**Note:** Note that domain model is used for write methods *POST* (*Create*) and *PUT* (*Update*) and as result of read-only method *GET/:id* (*GetById*).

| Name | Description | Type | Required | Read-only | Default |
|---|---|---|---|---|---|
| `id` | Representing **Team** identifier. | `guid` | yes | yes | |
| `name` | The name of **Team**. | `string(50)` | yes | no | |
| `description` | The description of **Team**. | `string(50)` | no | no | |
| `isActive` | Denotes whether the **Team** state is active or inactive. | `boolean` | yes | no | active (`true`) |
| `isDeleted` | Denotes whether the **Team** state is deleted or not. | `boolean` | yes | [partially] | not deleted (`false`) |
| `unitId` | Represent the belonging *Unit*. | `guid` | yes | no | |
| `showScore` | Displays percentage score in *QM Evaluation* while is being created instead of default when *Evaluation* is completed. | `boolean` | yes | no | don't show (`false`) |
| `managers` | The **Team** assigned [Managers (User)](/v1/user). | `array(guid)` | no | no | |
| `members` | The **Team** assigned [Members (User)](/v1/user). | `array(guid)` | no | no | |
| **Lookups** | | | | | |
| `unitsLookup` | The lookup dictionary of active and not deleted *Unit*, needed for choosing belonging *Unit* and setting `unitId` | `dictionary(guid, string)` | N/A | N/A | N/A |
| `usersLookup` | The dictionary of active and not deleted *Users*, needed for setting the **Team** `managers` and `members`. | `dictionary(guid, string)` | N/A | N/A | N/A |

**Note:** The **Team** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Team**

properties are capitalized (eg. `Id`, `Name`,..)!

**Warning:**
**Active and Deleted Logic**

When **Team** is deleted by [Delete command](/v1/team#team-delete) and it is flagged as `isDeleted` as `true` and also by default it is flagged `isActive` as `false`. Note that status `isActive` will remain "*locked*" until the **Team's** `isDeleted` state is updated to `false` or *not deleted* anymore. Then `isActive` is "*unlocked*" and can be changed. If the **Team** is *deleted* and on update is tried to change `isActive` property, server will silently ignore sent `isActive` property.

[Team Manager(s)](/v1/user), [Team Agent(s)](/v1/user) and [belonging Unit](/v1/unit) can be set to only *active* and *not deleted* **Teams**. If is sent otherwise to *inactive* and/or *deleted* **Team**, server will silently ignore those assignments.

## 8.2 Team List Model

Represent the **Team** list model with available properties.

**Note:**

The list model used only to list **Teams** with *GET* (*GetAll*) method.
Note that list model can change by adding/removing properties depending what users of *Coach REST API* will need in future.

| Name | Description | Type |
|---|---|---|
| `id` | Representing **Team** identifier. | `guid` |
| `name` | The name of **Team**. | `string` |
| `description` | The description of **Team**. | `string` |
| `isActive` | Denotes whether the **Team** state is active or inactive. | `boolean` |
| `isDeleted` | Denotes whether the **Team** state is deleted or not. | `boolean` |
| `unitName` | The name of belonging *Unit*. Needed for representing the parent unit in list of Team. | `string` |

**Note:** The **Team** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Team** properties are capitalized (eg. `Id`, `Name`,..)!

## 8.3 List of Teams

The list of **Teams** for current *Tenant*.

### 8.3.1 Default REST approach

GET /api/v1/:tenantCode/teams

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.

> **Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` array of [Team List Model](/v1/team#team-list-model).
- If there is an error: `JSON` *Client Errors* object.

### 8.3.2 C# Wrapper approach

```
TeamWrapper(int tenantCode, string apiKey, string apiSecret).GetAll();
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.

#### Return value

- If there is no error: `ResaultContent<ICollection<Team>>.Result` object collection of the [Team List Model](/v1/team#team-list-model).
- If there is an error: `ResaultContent<ICollection<Team>>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<Team, TeamList> teamWrapper = new TeamWrapper(tenantCode, key, secret);
ResponseContent<ICollection<TeamList>> response = teamWrapper.GetAll();

if (response.Result != null)
{
     // Use Result as List of Teams for displaying.
     ICollection<TeamList> teams = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 8.4 Get Team by Id

The **Team** by requested Id for current *Tenant*.

### 8.4.1 Default REST approach

GET /api/v1/:tenantCode/teams/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Team** id, a valid and non-empty `guid`.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` as the [Team Domain Model](/v1/team#team-model) object.
- If there is an error: `JSON` as the *Client Errors* object.

### 8.4.2 C# Wrapper approach

```
TeamWrapper(int tenantCode, string apiKey, string apiSecret).GetById(Guid id);
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `id` The **Team** id, a valid and non-empty `guid`.

#### Return value

- If there is no error: `ResaultContent<Team>.Result` object as the [Team Domain Model](/v1/team#team-model).
- If there is an error: `ResaultContent<Team>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid teamId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<Team, TeamList> teamWrapper = new TeamWrapper(tenantCode, key, secret);
ResponseContent<Team> response = teamWrapper.GetById(teamId);

if (response.Result != null)
{
     // Use Result as requested Team for displaying.
     Team team = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 8.5 Create Team

The creation of new **Team** for current *Tenant*.

### 8.5.1 Default REST approach

POST /api/v1/:tenantCode/teams

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `team` `JSON` representation of **Team properties** sent via *Request HTTP Header*.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` representation of newly created **Team** as the [Team Domain Model](/v1/team#team-model).
- If there is an error: `JSON` *Client Errors* object.

### 8.5.2 C# Wrapper approach

```
TeamWrapper(int tenantCode, string apiKey, string apiSecret).Create(Team team);
```

### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `team` The **Team** model constructed from **Team properties**.

### Return value

- If there is no error: `ResaultContent<Team>.Result` object as the [Team Domain Model](/v1/team#team-model).
- If there is an error: `ResaultContent<Team>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<Team, TeamList> teamWrapper = new TeamWrapper(tenantCode, key, secret);
// Get default data and lookup for teams
Team newTeam = teamWrapper.GetById(new Guid()).Result;
newTeam.Name = "Team created from test";
newTeam.Description = "Team created from test description.";
// Set belonging Unit key from units lookup key.
newTeam.UnitId = newTeam.Units.FirstOrDefault().Key;
ResponseContent<Team> response = teamWrapper.Create(newTeam);

if (response.Result != null)
{
     // Use Result as newly created Team for display.
     Team team = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 8.6 Update Team

Updates already existent **Team** for current *Tenant*.

### 8.6.1 Default REST approach

PUT /api/v1/:tenantCode/teams/:id

### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Team** id, a valid and non-empty `guid`.
- `team` `JSON` representation of **Team properties** sent via *Request HTTP Header*.

> **Danger:**
> Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*. \
> If you don't want to have in Web Server turned on the `PUT` verb method read more in *Getting Started*.

### Return value

- If there is no error: `JSON` representation of uodated **Team** as the [Team Domain Model](/v1/team#team-model) object.
- If there is an error: `JSON` *Client Errors* object.

## 8.6.2 C# Wrapper approach

```
TeamWrapper(int tenantCode, string apiKey, string apiSecret).Update(Team team, bool updateViaPost = f
```

### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `team` The **Team** model constructed from **Team properties** and `Id` must be provided in it. If not `ArgumentException` will be thrown!
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `PUT` method. Default is `false` or use `PUT` method!

### Return value

- If there is no error: `ResaultContent<Team>.Result` object as the [Team Domain Model](/v1/team#team-model).
- If there is an error: `ResaultContent<Team>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid teamId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<Team, TeamList> teamWrapper = new TeamWrapper(tenantCode, key, secret);
```

```
Team team = teamWrapper.GetById(teamId).Result;
team.Name = "Team updated from test";
team.Description = "Team updated from test description.";
// Set belonging Unit key from units lookup key.
team.UnitId = team.Units.FirstOrDefault().Key;

// Update via PUT method (default).
ResponseContent<Team> response = teamWrapper.Update(team);

// Update via POST method (use true argument).
// ResponseContent<Team> response = teamWrapper.Update(team, true);

if (response.Result != null)
{
     // Use Result of updated Team for display.
     Team updatedTeam = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 8.7 Delete Team

Deletes existent **Team** for current *Tenant*.

> **Warning:**
> Note that if **Team** has assigned *Team Managers* and *Agents* then **Team** will not be deleted but flagged as `isDeleted`. When **Team** is deleted it can be undeleted by setting `isDeleted` to `false` while updating **Team**. \
> If **Team** has no assigned *Team Managers* and *Agents*, it will be deleted permanently.

### 8.7.1 Default REST approach

DELETE /api/v1/:tenantCode/teams/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Team** id, a valid and non-empty `guid`.

> **Danger:**
> Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*. \
> If you don't want to have in Web Server turned on the `DELETE` verb method read more in *Getting Started*.

#### Return value

- There is no return value except if there is an error, the `JSON` *Client Errors* object.

### 8.7.2 C# Wrapper approach

```
TeamWrapper(int tenantCode, string apiKey, string apiSecret).Delete(Guid id, bool updateViaPost = fal
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `id` The **Team** id, a valid and non-empty `guid`.
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `DELETE` method. Default is `false` or use `DELETE` method!

#### Return value

- If there is no error: no return value or `void`.
- If there is an error: `ResaultContent<Team>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid teamId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<Team, TeamList> teamWrapper = new TeamWrapper(tenantCode, key, secret);
// Delete via DELETE method (default).
ResponseContent response = teamWrapper.Delete(teamId);

// Delete via POST method (use true argument).
// ResponseContent response = teamWrapper.Delete(teamId, true);

if (response.Error != null)
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

CHAPTER 9

# User

A **User** is an individual within the organization. They can be assigned any *Role* but most will be designated as an *Agent* or as the *Manager* of a *Team* and/or *Unit*. A **User** can be n *Manager*, *Agent* or *unassigned*. The *unassigned* users are not visible in hierarchical tree, see more in [Tenant Tree](/v1/tree).

**Note:** The number of **Users** that can be added to a *Tenant* is not limited but the number that can be activated is restricted to the number of licenses that have been purchased. For more information see [Licensing](/v1/licensing).

## 9.1 User Domain Model

Represent the **User** domain model with available properties and its behaviors.

**Note:** Note that domain model is used for write methods *POST* (*Create*) and *PUT* (*Update*) and as result of read-only method *GET/:id* (*GetById*).

| Name | Description |
|---|---|
| `id` | Representing **User** identifier. |
| **Personal Details** | |
| `username` | The username of **User**. |
| `firstName` | The **User** first name. |
| `lastName` | The **User** last name. |
| `email` | The **User** email. |
| `phone` | The **User** phone number. |
| `address` | The **User** residing address. |
| `country` | The **User** residing country. |
| `dateOfBirth` | The **User** date of birth. |
| `isActive` | Denotes whether the **User** state is active or inactive. |
| `isDeleted` | Denotes whether the **User** state is deleted or not. |
| **Employee Details** | |
| `employeeReference` | E.g. Payroll number or another unique identifier. |
| `startDate` | The **User** employment start date. |
| `endDate` | The **User** employment end date. |
| **Recorder** | |
| `recorderPlayerId` | The [Recorder](/v1/unit) where the **User's** media files are recorded and stored. |
| `recorderUserId` | The **User** identification of media files within the [Recorder](/v1/recorder). |
| `recorderAccountId` | |

Table 9.1 – continue

| Name | Description |
|---|---|
| **User Managerships,** | **Memberships and Roleships** |
| `managedUnits` | The **User's** managed [Units](/v1/unit). |
| `managedTeams` | The **User's** managed [Teams](/v1/team). |
| `teamMemberships` | The **User's** *Team* memberships. |
| `roles` | The **User's** assigned *Roles* |
| **Lookups** | |
| `unitsLookup` | The dictionary of active and not deleted [Units](/v1/unit), needed for setting the `managedU` |
| `teamsLookup` | The dictionary of active and not deleted [Teams](/v1/team), needed for setting the **User's** `ma` |
| `rolesLookup` | The dictionary of only adminstrative roles, needed for setting the **User's** `roles`. Read mor |
| `recorderMediaPlayersLookup` | The dictionary of active [Recorders](/v1/recorder), needed for setting the **User's** `recorerl` |

**Note:** The **User** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **User** properties are capitalized (eg. `Id`, `Name`,..)!

**Warning:**
**Active and Deleted Logic**

When **User** is deleted by [Delete](/v1/user#user-delete) command and it is flagged as `isDeleted` as `true` and also by default it is flagged `isActive` as `false`. Note that status `isActive` will remain "*locked*" until the **User's** `isDeleted` state is updated to `false` or *not deleted* anymore. Then `isActive` is "*unlocked*" and can be changed. If the **User** is *deleted* and on update is tried to change `isActive` property, server will silently ignore sent `isActive` property.

Managed [Unit(s)](/v1/unit), Managed [Team(s)](/v1/team) and [Administrative Role(s)](/v1/roles) can be set to only *active* and *not deleted* **Users**. If is sent otherwise to *inactive* and/or *deleted* **User**, server will silently ignore those assignments.

## 9.2 User List Model

Represent the **User** list model with available properties.

**Note:**

The list model used only to list **Users** with *GET* (*GetAll*) method.

Note that list model can change by adding/removing properties depending what users of *Coach REST API* will need in future.

| Name | Description | Type |
|---|---|---|
| `id` | Representing **User** identifier. | `guid` |
| `username` | The username of **User**. | `string` |
| `firstName` | The **User** first name. | `string` |
| `lastName` | The **User** last name. | `string` |
| `isActive` | Denotes whether the **User** state is active or inactive. | `boolean` |
| `isDeleted` | Denotes whether the **User** state is deleted or not. | `boolean` |
| `recorderUserId` | The **User** identification of media files within the Recorder. | `string` |
| `recorderAccountId` | | `string(50)` |
| `managedUnits` | The **User's** managed *Units* comma separated. | `string` |
| `managedTeams` | The **User's** managed *Teams* comma separated. | `string` |
| `teamMemberships` | The **User's** *Team* memberships comma separated. | `string` |
| `assignedRoles` | The **User's** assigned *Roles* comma separated. | `string` |

**Note:** The **User** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **User** properties are capitalized (eg. `Id`, `Name`,..)!

## 9.3 List of Users

The list of **Users** for current *Tenant*.

### 9.3.1 Default REST approach

GET /api/v1/:tenantCode/users

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` array of [User List Model](/v1/user#user-list-model).
- If there is an error: `JSON` *Client Errors* object.

### 9.3.2 C# Wrapper approach

```
1 UserWrapper(int tenantCode, string apiKey, string apiSecret).GetAll();
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.

### Return value

- If there is no error: `ResaultContent<ICollection<User>>.Result` object as collection of the [User List Model](/v1/user#user-list-model).
- If there is an error: `ResaultContent<ICollection<User>>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<User, UserList> userWrapper = new UserWrapper(tenantCode, key, secret);
ResponseContent<ICollection<UserList>> response = userWrapper.GetAll();

if (response.Result != null)
{
     // Use Result as List of Users for displaying.
     ICollection<UserList> users = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 9.4 Get User by Id

The **User** by requested Id for current *Tenant*.

### 9.4.1 Default REST approach

GET /api/v1/:tenantCode/users/:id

### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **User** id, a valid and non-empty `guid`.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

### Return value

- If there is no error: `JSON` as the [User Domain Model](/v1/user#user-model) object.
- If there is an error: `JSON` as the *Client Errors* object.

### 9.4.2 C# Wrapper approach

```
UserWrapper(int tenantCode, string apiKey, string apiSecret).GetById(Guid id);
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `id` The **User** id, a valid and non-empty `guid`.

#### Return value

- If there is no error: `ResaultContent<User>.Result` object as the [User Domain Model](/v1/user#user-model).
- If there is an error: `ResaultContent<User>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid userId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<User, UserList> userWrapper = new UserWrapper(tenantCode, key, secret);
ResponseContent<User> response = userWrapper.GetById(userId);

if (response.Result != null)
{
     // Use Result as requested User for displaying.
     User user = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 9.5 Create User

The creation of new **User** for current *Tenant*.

### 9.5.1 Default REST approach

POST /api/v1/:tenantCode/users

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `user` `JSON` representation of [User Domain Model](/v1/user#user-model) sent via *Request HTTP Header*.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` representation of newly created **User** as the [User Domain Model](/v1/user#user-model).
- If there is an error: `JSON` *Client Errors* object.

### 9.5.2 C# Wrapper approach

```
UserWrapper(int tenantCode, string apiKey, string apiSecret).Create(User user);
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `user` The **User** model constructed from **User properties**.

#### Return value

- If there is no error: `ResaultContent<User>.Result` object as the [User Domain Model](/v1/user#user-model).
- If there is an error: `ResaultContent<User>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<User, UserList> userWrapper = new UserWrapper(tenantCode, key, secret);
// Get default data and lookup for users
User newUser = userWrapper.GetById(new Guid()).Result;
newUser.Username = "Tester";
newUser.RecorderMediaPlayerId = newUser.RecorderMediaPlayersLookup.FirstOrDefault().Key;
newUser.Roles = new List<Guid> { newUser.RolesLookup.FirstOrDefault().Key };
newUser.ManagedUnits = new List<Guid> { newUser.UnitsLookup.FirstOrDefault().Key };
newUser.ManagedTeams = new List<Guid> { newUser.TeamsLookup.FirstOrDefault().Key };
newUser.TeamMemberships = new List<Guid> { newUser.TeamsLookup.LastOrDefault().Key };

```

```
ResponseContent<User> response = userWrapper.Create(newUser);

if (response.Result != null)
{
     // Use Result as newly created User for display.
     User user = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 9.6 Update User

Updates already existent **User** for current *Tenant*.

### 9.6.1 Default REST approach

PUT /api/v1/:tenantCode/users/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **User** id, a valid and non-empty `guid`.
- `user` `JSON` representation of [User Domain Model](/v1/user#user-model) sent via *Request HTTP Header*.

**Danger:**
Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*.
See more in *Getting Started*.
If you don't want to have in Web Server turned on the `PUT` verb method read more in *Getting Started*.

#### Return value

- If there is no error: `JSON` representation of uodated **User** as the [User Domain Model](/v1/user#user-model).
- If there is an error: `JSON` *Client Errors* object.

### 9.6.2 C# Wrapper approach

```
UserWrapper(int tenantCode, string apiKey, string apiSecret).Update(User user, bool updateViaPost = f
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.

- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `user` The **User** model constructed from **User properties** and `Id` must be provided in it. If not `ArgumentException` will be thrown!
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `PUT` method. Default is `false` or use `PUT` method!

### Return value

- If there is no error: `ResaultContent<User>.Result` object as the [User Domain Model](/v1/user#user-model).
- If there is an error: `ResaultContent<User>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid userId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<User, UserList> userWrapper = new UserWrapper(tenantCode, key, secret);
User user = userWrapper.GetById(userId).Result;
user.Username = "Tester";
user.RecorderMediaPlayerId = user.RecorderMediaPlayersLookup.FirstOrDefault().Key;
user.Roles = new List<Guid> { user.RolesLookup.FirstOrDefault().Key };
user.ManagedUnits = new List<Guid> { user.UnitsLookup.FirstOrDefault().Key };
user.ManagedTeams = new List<Guid> { user.TeamsLookup.FirstOrDefault().Key };
user.TeamMemberships = new List<Guid> { user.TeamsLookup.LastOrDefault().Key };

// Update via PUT method (default).
ResponseContent<User> response = userWrapper.Update(user);

// Update via POST method (use true argument).
// ResponseContent<User> response = userWrapper.Update(user, true);

if (response.Result != null)
{
     // Use Result of updated User for display.
     User updatedUser = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 9.7 Delete User

Deletes existent **User** for current *Tenant*.

**Warning:**
Note that if **User** is a *Unit Manager*, *Team Manager*, *Agent* or has some [Role](/v1/roles) assigned to it then **User** will not be deleted but flagged as `isDeleted`. When **User** is deleted it can be undeleted by setting `isDeleted` to `false` while updating **Unit**.
If **User** is not a *Unit Manager*, *Team Manager*, *Agent* or has no [Role](/v1/roles) assigned to it, it will be deleted permanently.

### 9.7.1 Default REST approach

DELETE /api/v1/:tenantCode/users/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **User** id, a valid and non-empty `guid`.

**Danger:**
Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.
If you don't want to have in Web Server turned on the `DELETE` verb method read more in *Getting Started*.

#### Return value

- There is no return value except if there is an error, the `JSON` *Client Errors* object.

### 9.7.2 C# Wrapper approach

```
UserWrapper(int tenantCode, string apiKey, string apiSecret).Delete(Guid id, bool updateViaPost = fal
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `id` The **User** id, a valid and non-empty `guid`.
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `DELETE` method. Default is `false` or use `DELETE` method!

#### Return value

- If there is no error: no return value or `void`.
- If there is an error: `ResaultContent<User>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid userId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<User, UserList> userWrapper = new UserWrapper(tenantCode, key, secret);
// Delete via DELETE method (default).
ResponseContent response = userWrapper.Delete(userId);

// DELETE via POST method (use true argument)..
// ResponseContent response = userWrapper.Delete(userId, true);

if (response.Error != null)
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

CHAPTER 10

# Media Player

**Media Player** describes the way the media files will be played in *Coach*.

## 10.1 Media Player Types

### 10.1.1 Default Internal

The Default Internal *Coach* Player is a Silverlight player and is to be used with those [Recorders](/v1/recorder) that use codecs compatible with Silverlight.

### 10.1.2 Custom Internal

The Custom Internal Player should be selected if you, the customer, will be using your own web-based player, or that of a third party, that meets a strict API compliance.

**Warning:** The Internet Explorer browser uses Windows Media Player and Firefox / Chrome uses QuickTime. It can be used any media player as long as it has an API that meets *Coach* playback contract. The player's API must consist of the following API: Load (URL or ID), Play, Stop, Play, Seek from seconds, to seconds) and finally Get duration of recording in seconds. The player's API must be accessible via javascript.

### 10.1.3 External

This is when the player is contained in the user's own application and *Coach* provides only recording playback details.

### 10.1.4 None

*Coach* will provide no playback or playback details for recordings.

## 10.2 Media Player Domain Model

Represent the **Media Player** domain model with available properties and its behaviors.

**Note:** Note that domain model is used for write methods *POST* (*Create*) and *PUT* (*Update*) and as result of read-only

method *GET/:id* (*GetById*).

| Name | Description | Type | Required | Read-only | Default |
|---|---|---|---|---|---|
| `id` | Representing **Media Player** identifier. | `guid` | yes | yes | |
| `name` | The name of **Media Player**. | `string(50)` | yes | no | |
| `type` | The [type](/v1/player#media-player-types) of **Media Player**. | `byte` | yes | no | Default internal (`1`) |
| `isAutoPlay` | Denotes whether the media file will start instantly. | `bool` | no | no | `false` |
| `publishingPoint` | The domain name of the Windows Media Service publishing point of the root from where the media files are hosted. URL can be relative or absolute. | `string(500)` | no | no | |
| `playerUrl` | URL of HTML file that contains the **Media Player**. URL can be relative or absolute. | `string(500)` | no | no | |
| `rootFolder` | The URL of root folder from where the media files are hosted. URL can be relative or absolute. | `string(500)` | no | no | |
| `play` | The Javascript function used to Play the media. | `string(50)` | no | no | |
| `stop` | The Javascript function used to Stop the media. | `string(50)` | no | no | |
| `pause` | The Javascript function used to Pause the media. | `string(50)` | no | no | |
| `seek` | The Javascript function used to Seek the media. | `string(50)` | no | no | |
| `load` | The Javascript function used to Load the media. | `string(50)` | no | no | |
| `length` | The Javascript function used to return the length in seconds of the media. | `string(50)` | no | no | |
| `lastSaved` | The date the **Media Player** was last saved. | `datetime` | no | no | `null` |
| `loadType` | The [load type](/v1/player#load-types) of **Media Player**. | `byte` | no | no | None (`2`) |
| **Lookups** | | | | | |
| `types` | The lookup dictionary of [Media Player Types](/v1/player#media-player-types). Used to set the `playerType`. | `dictionary(byte, string)` | N/A | N/A | N/A |
| `loadTypes` | The lookup dictionary of [Media Player Load Types](/v1/player#load-types). Used to set the `loadType`. | `dictionary(byte, string)` | N/A | N/A | N/A |

**Note:** The **Media Player** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Media Player** properties are capitalized (eg. `Id`, `Name`,..)!

### 10.2.1 Media Player Required Properties By Player Type

The table of required Domain Model properties depending on Media Player Type

| Player Type | `publishingPoint` | `isAutoPlay` | `playerUrl` | `rootFolder` | `play` | `stop` | `pause` | `stop` | `seek` | `length` | `loadType` |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Default Internal** | | | | | | | | | | | |
| **Custom Internal** | | | | | | | | | | | |
| **External** | | N/A | | | | | | | | | |
| **None** | | N/A | | | | | | | | | |

### 10.2.2 Media Player Load Types

It is the norm for a browser based player like WMP to accept an URL. However, in the cases where the media file has been encrypted, some propriety players insist on an ID. This ID will use to source the precise media file.

#### URL

Load URL into media player.

#### Id

Load ID into media player.

#### None

Do not load anything into media player.

## 10.3 Media Player List Model

Represent the **Media Player** list model with available properties.

---

**Note:**

The list model used only to list **Media Players** with *GET* (*GetAll*) method.

Note that list model can change by adding/removing properties depending what users of *Coach REST API* will need in future.

---

| Name | Description | Type |
|---|---|---|
| `id` | Representing **Media Player** identifier. | `guid` |
| `name` | The name of **Media Player**. | `string` |
| `playerTypeName` | The type of **Media Player**. | `string` |
| `lastSaves` | The date when the **Media Player** was last saved. | `datetime` |

**Note:** The **Media Player** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Media Player** properties are capitalized (eg. `Id`, `Name`,..)!

---

## 10.4 List of Media Players

The list of **Media Players** for current *Tenant*.

### 10.4.1 Default REST approach

GET /api/v1/:tenantCode/players

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` array of [Media Player List Model](/v1/player#media-player-list-model).
- If there is an error: `JSON` *Client Errors* object.

### 10.4.2 C# Wrapper approach

```
MediaPlayerWrapper(int tenantCode, string apiKey, string apiSecret).GetAll();
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.

#### Return value

- If there is no error: `ResaultContent<ICollection<MediaPlayer>>.Result` object collection of the [Media Player List Model](/v1/player#media-player-list-model).
- If there is an error: `ResaultContent<ICollection<MediaPlayer>>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<MediaPlayer, MediaPlayerList> mediaPlayerWrapper = new MediaPlayerWrapper(tenantCode,
ResponseContent<ICollection<MediaPlayerList>> response = mediaPlayerWrapper.GetAll();

if (response.Result != null)
{
     // Use Result as List of Media Players for displaying.
     ICollection<MediaPlayerList> mediaPlayers = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 10.5 Get Media Player by Id

The **Media Player** by requested Id for current *Tenant*.

### 10.5.1 Default REST approach

GET /api/v1/:tenantCode/players/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Media Player** id, a valid and non-empty `guid`.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` as the [Media Player Domain Model](/v1/player#media-player-model) object.
- If there is an error: `JSON` as the *Client Errors* object.

### 10.5.2 C# Wrapper approach

```
MediaPlayerWrapper(int tenantCode, string apiKey, string apiSecret).GetById(Guid id);
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `id` The **Media Player** id, a valid and non-empty `guid`.

#### Return value

- If there is no error: `ResaultContent<MediaPlayer>.Result` object as the [Media Player Domain Model](/v1/player#media-player-model).
- If there is an error: `ResaultContent<MediaPlayer>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid mediaPlayerId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<MediaPlayer, MediaPlayerList> mediaPlayerWrapper = new MediaPlayerWrapper(tenantCode,
ResponseContent<MediaPlayer> response = mediaPlayerWrapper.GetById(mediaPlayerId);

if (response.Result != null)
{
     // Use Result as requested Media Player for displaying.
     MediaPlayer mediaPlayer = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 10.6 Create Media Player

The creation of new **Media Player** for current *Tenant*.

### 10.6.1 Default REST approach

POST /api/v1/:tenantCode/players

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `mediaPlayer` `JSON` representation of **Media Player properties** sent via *Request HTTP Header*.

**Danger:** Remember to add *API Key* as *customer key* and *API Secret* as *customer secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` representation of newly created **Media Player** as the [Media Player Domain Model](/v1/player#media-player-model).
- If there is an error: `JSON` *Client Errors* object.

### 10.6.2 C# Wrapper approach

```
MediaPlayerWrapper(int tenantCode, string apiKey, string apiSecret).Create(MediaPlayer mediaPlayer);
```

### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `mediaPlayer` The **Media Player** model constructed from **Media Player properties**.

### Return value

- If there is no error: `ResaultContent<MediaPlayer>.Result` object as the [Media Player Domain Model](/v1/player#media-player-model).
- If there is an error: `ResaultContent<MediaPlayer>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<MediaPlayer, MediaPlayerList> mediaPlayerWrapper = new MediaPlayerWrapper(tenantCode,
// Get default data and lookup for media players
MediaPlayer newMediaPlayer = mediaPlayerWrapper.GetById(new Guid()).Result;
newMediaPlayer.Name = "Media Player created from test";

// Set type from media types lookup key.
newMediaPlayer.Type = newMediaPlayer.Types.FirstOrDefault().Key;
newMediaPlayer.PublishingPoint = "xyz";
ResponseContent<MediaPlayer> response = mediaPlayerWrapper.Create(newMediaPlayer);

if (response.Result != null)
{
     // Use Result as newly created Media Player for display.
     MediaPlayer mediaPlayer = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 10.7 Update Media Player

Updates already existent **Media Player** for current *Tenant*.

### 10.7.1 Default REST approach

PUT /api/v1/:tenantCode/players/:id

### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Media Player** id, a valid and non-empty `guid`.
- `mediaPlayer` `JSON` representation of **Media Player properties** sent via *Request HTTP Header*.

**Danger:**
Remember to add *API Key* as *customer key* and *API Secret* as *customer secret* into your *Request HTTP Header*. See more in *Getting Started*.
If you don't want to have in Web Server turned on the `PUT` verb method read more in *Getting Started*.

### Return value

- If there is no error: `JSON` representation of uodated **Media Player** as the [Media Player Domain Model](/v1/player#media-player-model) object.
- If there is an error: `JSON` *Client Errors* object.

## 10.7.2 C# Wrapper approach

```
MediaPlayerWrapper(int tenantCode, string apiKey, string apiSecret).Update(MediaPlayer mediaPlayer, b
```

### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `mediaPlayer` The **Media Player** model constructed from **Media Player properties** and `Id` must be provided in it. If not `ArgumentException` will be thrown!
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `PUT` method. Default is `false` or use `PUT` method!

### Return value

- If there is no error: `ResaultContent<MediaPlayer>.Result` object as the [Media Player Domain Model](/v1/player#media-player-model).
- If there is an error: `ResaultContent<MediaPlayer>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid mediaPlayerId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<MediaPlayer, MediaPlayerList> mediaPlayerWrapper = new MediaPlayerWrapper(tenantCode,
```

```
MediaPlayer mediaPlayer = mediaPlayerWrapper.GetById(mediaPlayerId).Result;
mediaPlayer.Name = "Media Player updated from test";
mediaPlayer.PublishingPoint = "xyz updated";

// Update via PUT method (default).
ResponseContent<MediaPlayer> response = mediaPlayerWrapper.Update(mediaPlayer);

// Update via POST method (use true argument).
// ResponseContent<MediaPlayer> response = mediaPlayerWrapper.Update(mediaPlayer, true);

if (response.Result != null)
{
     // Use Result of updated Media Player for display.
     MediaPlayer updatedMediaPlayer = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 10.8 Delete Media Player

Deletes existent **Media Player** for current *Tenant*.

> **Warning:** Note that **Media Player** will not be deleted if there are any references of **Media Player** in [Recorder Media Player](/v1/recorder-player).

### 10.8.1 Default REST approach

DELETE /api/v1/:tenantCode/players/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Media Player** id, a valid and non-empty `guid`.

> **Danger:**
> Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.
> If you don't want to have in Web Server turned on the `DELETE` verb method read more in *Getting Started*.

#### Return value

- There is no return value except if there is an error, the `JSON` *Client Errors* object.

### 10.8.2 C# Wrapper approach

```
MediaPlayerWrapper(int tenantCode, string apiKey, string apiSecret).Delete(Guid id, bool updateViaPos
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `id` The **Media Player** id, a valid and non-empty `guid`.
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `DELETE` method. Default is `false` or use `DELETE` method!

#### Return value

- If there is no error: no return value or `void`.
- If there is an error: `ResaultContent<MediaPlayer>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid mediaPlayerId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<MediaPlayer, MediaPlayerList> mediaPlayerWrapper = new MediaPlayerWrapper(tenantCode,
// Delete via DELETE method (default).
ResponseContent response = mediaPlayerWrapper.Delete(mediaPlayerId);

// Delete via POST method (use true argument).
// ResponseContent response = mediaPlayerWrapper.Delete(mediaPlayerId, true);

if (response.Error != null)
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

CHAPTER 11

# Recorder

The **Recorder** is set of settings to integrate the media **Recorder** with *Coach* for playing media files and getting media files for *Evaluation*.

## 11.1 Recorder Components

The **Recorder** components are: *Recorders*, *Media Players*, *Recorder Media Players*.

### 11.1.1 Recorders

A set of configuration settings that will connect **Recorder(s)** with *Coach*.

### 11.1.2 Media Players

Describes the way the media files will be played in *Coach*.

### 11.1.3 Recorder Media Players

Here a **Recorder** that has been configured with *Coach* is combined with a **Media Player** that has been entered so that all media files evaluated in *Coach QM* from this **Recorder** will be replayed using this set **Media Player**.

## 11.2 Recorder Domain Model

Represent the **Recorder** domain model with available properties and its behaviors.

**Note:** Note that domain model is used for write methods *POST* (*Create*) and *PUT* (*Update*) and as result of read-only method *GET/:id* (*GetById*).

| Name | Description | Type | Required | Read-only | Default |
|---|---|---|---|---|---|
| `id` | Representing **Recorder** identifier. | `guid` | yes | yes | |
| `name` | The name of **Recorder**. | `string(50)` | yes | no | |
| `databaseServerIP` | The **Recorder's** IP / domain name of SQL Server. | `string(1000)` | no | no | |
| `databaseName` | The **Recorder** database name that will be queried for media files. | `string(1000)` | no | no | |
| `databaseUsername` | The username of the DB login that will be used to gain access to the database. | `string(1000)` | no | no | |
| `databasePassword` | The password of the DB login that will be used to gain access to the database. | `string(1000)` | no | no | |
| `databasePort` | The **Recorder's** database port. | `string(6)` | no | no | |
| `serverIP` | The **Recorder** IP / domain name of where the database service is hosted. | `string(1000)` | yes | no | |
| `tableName` | The **Recorder** DB table name where media files are persisted. | `string(1000)` | yes | no | |
| `communicationMethodType` | The communication method](/v1/recorder#communication-method) between *Coach* and the **Recorder**. | `byte` | yes | no | Direct Access `1` |
| `databaseType` | The supported DB engines for getting **Recorder's** media files. | `byte` | yes | no | SqlServer `1` |
| `searchCriteriaCollection` | The collection of [Search Criteria](/v1/search-criteria). | `array(SearchCriteria)` | no | no | |
| `mediaFileMetadataCollection` | The **Recorder** media metadata for media files DB properties](/v1/media-metadata). | `array(MediaFileMetadata)` | yes | partially](/v1/recorder#media-file-metadata) | |
| **Lookups** | | | | | |
| `communicationMethodTypes` | The dictionary of [Communication Method Types](/v1/recorder#communication-method) for `communicationMethodType`. | `dictionary(byte, string)` | N/A | N/A | N/A |
| `databaseTypes` | The dictionary of supported [Database Engine Types](/v1/player#database-type) for `databaseType`. | `dictionary(byte, string)` | N/A | N/A | N/A |
| `databaseDataTypes` | The dictionary of supported [Database Data Types](/v1/player) for `databaseType`. | `dictionary(byte, string)` | N/A | N/A | N/A |
| `searchDataTypesWithConditions` | The collection of [Data Types and its supported Conditions](/v1/search-criteria#data-condition-model) | `array(SearchDataTypeWithConditions)` | N/A | N/A | N/A |

**Note:** The **Recorder** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Recorder** properties are capitalized (eg. `Id`, `Name`,..)!

**Danger:** The *SQL Server* uses by default port 1433 and is not needed. If *MySQL* database is used the database port is required.

When the particular **Recorder** is acquired by *REST API*, for `mediaFileMetadataCollection` will be sent as collection of [supported/required media file metadata](/v1/media-metadata#supported-metadata). \ Then there is only need for setting matching [Recorder's](/v1/recorder) `fieldName` and its [Database Data Type](/v1/media-metadata#database-data-types). \ Note that the `mediaFileMetadataCollection` is read-only collection, col-

lection items can only be changed, inserting or deleting items will result the exception from server. {: #media-file-metadata .alert .alert-error .alert-block }

### 11.2.1 Communication Method Types

#### Direct Access

**Warning:** This options is being phased out

Direct connection to a **Recorder's** database.

**Danger:** Note that if *Direct Access* is chosen than properties `databaseServerIP`, `databaseName`, `databaseUsername` and `databasePassword` are required!

#### API

Connection to a **Recorder's** repository when there is no direct connection available.

### 11.2.2 Database Types

The *Coach* supported database engines for getting the **Recorder's** media files.

Currently the *Microsoft SQL Server* and *Oracle MySQL* database engines are supported.

## 11.3 Recorder List Model

Represent the **Recorder** list model with available properties.

**Note:**

The list model used only to list **Recorders** with *GET* (*GetAll*) method.

Note that list model can change by adding/removing properties depending what recorders of *Coach REST API* will need in future.

| Name | Description \| Type | |
|---|---|---|
| `id` | Representing **Recorder** identifier. \| `guid` | |
| `name` | The name of **Recorder**. | `string` |
| `communicationMethodType` | The **Recorder** communication method type. | `string` |
| `databaseType` | The database engine type used for storing **Recorder** media files. | `string` |
| `serverIP` | The **Recorder** IP / domain name of where the database service is hosted. | `string` |
| `databaseServerIP` | The **Recorder's** IP / domain name of SQL Server. | `string` |

**Note:** The **Recorder** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Recorder** properties are capitalized (eg. `Id`, `Name`,..)!

## 11.4 List of Recorders

The list of **Recorders** for current *Tenant*.

### 11.4.1 Default REST approach

GET /api/v1/:tenantCode/recorders

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in [Getting Started](/v1).

#### Return value

- If there is no error: `JSON` array of [Recorder List Model](/v1/recorder#recorder-list-model).
- If there is an error: `JSON` *Client Errors* object.

### 11.4.2 C# Wrapper approach

```
RecorderWrapper(int tenantCode, string apiKey, string apiSecret).GetAll();
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.

#### Return value

- If there is no error: `ResaultContent<ICollection<Recorder>>.Result` object as collection of the [Recorder List Model](/v1/recorder#recorder-list-model).
- If there is an error: `ResaultContent<ICollection<Recorder>>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<Recorder, RecorderList> recorderWrapper = new RecorderWrapper(tenantCode, key, secret
ResponseContent<ICollection<RecorderList>> response = recorderWrapper.GetAll();
```

```

if (response.Result != null)
{
      // Use Result as List of Recorders for displaying.
      ICollection<RecorderList> recorders = response.Result;
}
else
{
     // TODO: The error handling...
     Console.WriteLine(response.Error);
}
```

## 11.5 Get Recorder by Id

The **Recorder** by requested Id for current *Tenant*.

### 11.5.1 Default REST approach

GET /api/v1/:tenantCode/recorders/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Recorder** id, a valid and non-empty `guid`.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in [Getting Started](/v1).

#### Return value

- If there is no error: `JSON` as the [Recorder Domain Model](/v1/recorder#recorder-model) object.
- If there is an error: `JSON` as the *Client Errors* object.

### 11.5.2 C# Wrapper approach

```
RecorderWrapper(int tenantCode, string apiKey, string apiSecret).GetById(Guid id);
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `id` The **Recorder** id, a valid and non-empty `guid`.

### Return value

- If there is no error: `ResaultContent<Recorder>.Result` object as the [Recorder Domain Model](/v1/recorder#recorder-model).
- If there is an error: `ResaultContent<Recorder>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid recorderId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<Recorder, RecorderList> recorderWrapper = new RecorderWrapper(tenantCode, key, secret
ResponseContent<Recorder> response = recorderWrapper.GetById(recorderId);

if (response.Result != null)
{
     // Use Result as requested Recorder for displaying.
     Recorder recorder = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 11.6 Create Recorder

The creation of new **Recorder** for current *Tenant*.

**Warning:** To assign `levels` you'll need to get [Tenant Tree](/v1/tree) and use [Unit, Team or Agent](/v1/tree#levels-and-item-types) items as Levels!

### 11.6.1 Default REST approach

POST /api/v1/:tenantCode/recorders

### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `recorder` `JSON` representation of [Recorder Domain Model](/v1/recorder#recorder-model) sent via *Request HTTP Header*.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in [Getting Started](/v1).

#### Return value

- If there is no error: `JSON` representation of newly created **Recorder** as the [Recorder Domain Model](/v1/recorder#recorder-model).
- If there is an error: `JSON` *Client Errors* object.

### 11.6.2 C# Wrapper approach

```
RecorderWrapper(int tenantCode, string apiKey, string apiSecret).Create(Recorder recorder);
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `recorder` The **Recorder** model constructed from **Recorder properties**.

#### Return value

- If there is no error: `ResaultContent<Recorder>.Result` object as the [Recorder Domain Model](/v1/recorder#recorder-model).
- If there is an error: `ResaultContent<Recorder>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<Recorder, RecorderList> recorderWrapper = new RecorderWrapper(tenantCode, key, secret
// Get default data and lookup for recorder
Recorder newRecorder = recorderWrapper.GetById(new Guid()).Result;
newRecorder.Name = "Test Recorder with Direct Access";
newRecorder.ServerIP = "localhost";
newRecorder.DatabaseServerIP = "localhost";
newRecorder.DatabaseName = "recordings";
newRecorder.DatabaseUsername = "user";
newRecorder.DatabasePassword = "$ecret";
newRecorder.CommunicationMethodType = newRecorder.CommunicationMethodTypes
                                                 .Where(x => x.Value == "DirectAccess")
                                                 .Select(x => x.Key)
                                                 .SingleOrDefault();

ResponseContent<Recorder> response = recorderWrapper.Create(newRecorder);

if (response.Result != null)
{
     // Use Result as newly created Recorder for display.
     Recorder recorder = response.Result;
}
```

```
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 11.7 Update Recorder

Updates already existent **Recorder** for current *Tenant*.

**Warning:** To assign `levels` you'll need to get [Tenant Tree](/v1/tree) and use [Unit, Team or Agent](/v1/tree#levels-and-item-types) items as Levels!

### 11.7.1 Default REST approach

PUT /api/v1/:tenantCode/recorders/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Recorder** id, a valid and non-empty `guid`.
- `recorder` `JSON` representation of [Recorder Domain Model](/v1/recorder#recorder-model) sent via *Request HTTP Header*.

**Danger:**
Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in [Getting Started](/v1).
If you don't want to have in Web Server turned on the `PUT` verb method read more in [Getting Started](/v1).

#### Return value

- If there is no error: `JSON` representation of uodated **Recorder** as the [Recorder Domain Model](/v1/recorder#recorder-model).
- If there is an error: `JSON` *Client Errors* object.

### 11.7.2 C# Wrapper approach

```
RecorderWrapper(int tenantCode, string apiKey, string apiSecret).Update(Recorder recorder, bool updat
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.

- `recorder` The **Recorder** model constructed from **Recorder properties** and `Id` must be provided in it. If not `ArgumentException` will be thrown!
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `PUT` method. Default is `false` or use `PUT` method!

### Return value

- If there is no error: `ResaultContent<Recorder>.Result` object as the [Recorder Domain Model](/v1/recorder#recorder-model).
- If there is an error: `ResaultContent<Recorder>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid recorderId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");


ITreeApiWrapper<Recorder, RecorderList> recorderWrapper = new RecorderWrapper(tenantCode, key, secret
Recorder recorder = recorderWrapper.GetById(recorderId).Result;

recorder.Name = "Test Recorder updated to API";
recorder.ServerIP = "localhost";
recorder.CommunicationMethodType = recorder.CommunicationMethodTypes
                                            .Where(x => x.Value == "API")
                                            .Select(x => x.Key)
                                            .SingleOrDefault();

// Update via PUT method (default).
ResponseContent<Recorder> response = recorderWrapper.Update(recorder);

// Update via POST method (use true argument).
// ResponseContent<Recorder> response = recorderWrapper.Update(recorder, true);

if (response.Result != null)
{
     // Use Result of updated Recorder for display.
     Recorder updatedRecorder = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 11.8 Delete Recorder

Deletes existent **Recorder** for current *Tenant*.

> **Warning:** Note that **Recorder** will not be deleted if there are any references of **Recorder** in [Recorder Media Player](/v1/recorder-player).

### 11.8.1 Default REST approach

DELETE /api/v1/:tenantCode/recorders/:id

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Recorder** id, a valid and non-empty `guid`.

> **Danger:**
> Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*.
> See more in [Getting Started](/v1).
> If you don't want to have in Web Server turned on the `DELETE` verb method read more in [Getting Started](/v1).

#### Return value

- There is no return value except if there is an error, the `JSON` *Client Errors* object.

### 11.8.2 C# Wrapper approach

```
RecorderWrapper(int tenantCode, string apiKey, string apiSecret).Delete(Guid id, bool updateViaPost =
```

#### Parameters

- `tenantCode` Current *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` Current *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` Current *Tenant* API Secret provided by **Qualtrak**.
- `id` The **Recorder** id, a valid and non-empty `guid`.
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `DELETE` method. Default is `false` or use `DELETE` method!

#### Return value

- If there is no error: no return value or `void`.
- If there is an error: `ResaultContent<Recorder>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid recorderId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<Recorder, RecorderList> recorderWrapper = new RecorderWrapper(tenantCode, key, secret
// Delete via DELETE method (default).
```

```
ResponseContent response = recorderWrapper.Delete(recorderId);

// DELETE via POST method (use true argument)..
// ResponseContent response = recorderWrapper.Delete(recorderId, true);

if (response.Error != null)
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

CHAPTER 12

# Search Criteria

The **search criteria** are the parameters that a *Manager* can use to search for individual recordings for evaluation in *Coach QM* and it is used for [Schedule](/v1/schedule). These parameters can be configured to reflect any type of search that can be performed within the *Recorder* itself, allowing for a seamless user experience between the integrated applications. As few or as many search criteria can be configured as required.

## 12.1 Search Criteria Model

Represents the **Search Criteria** model as value object with available properties and its behaviors used for *Recorder* and [Schedule](/v1/schedule).

| Name | Description | Type | Required | Read-only |
|---|---|---|---|---|
| `id` | Representing **Search Criteria** identifier. | `guid` | yes | yes |
| `name` | The label the criterion will be given in *QM*. | `string(30)` | yes | no |
| `fieldName` | The search field where is particular piece of information from e.g. date. | `string(30)` | yes | no |
| `dataType` | From the supported [Data Types](/v1/search-criteria#data-types). | `string` | yes | no |
| `conditionType` | From the supported [Condition Types](/v1/search-criteria#data-types). | `string` | yes | no |
| `listOptions` | Options collection of *key/values* for *List Data Type*. | `dictionary(string, string)` | no | no |
| `defaultValue` | The `listOptions` default selected value. | `string` | no | no |

**Note:** The **Search Criteria** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Search Criteria** properties are capitalized (eg. `Id`, `Name`,..)! \

## 12.2 Data Type with Conditions Model

Represents the *Data Type* with its supported *Conditions* as model for lookup with available properties used for setting valid combination of `dataType` and `conditionType` for **Search Criteria**.

| Name | Description | Type |
|---|---|---|
| `dateType` | The *Data Type* id and name. | `keyValue(byte, string)` |
| `conditions` | The *Data Type* supported *Conditions* as dictionary of id and name | `dictionary(byte, string)` |

**Note:**
The **Data Type with Condition** properties names (*Name* column) is
for default usage by `JSON`, for `C# Wrapper` usage the **Data Type**
with Condition** properties are capitalized (eg. `Id`, `Name`,..)!
There is metrics of available [Condition Types](/v1/search-criteria#data-types) for each [Data Type](/v1/search-criteria#data-types) see more in section ["Data type and Conditions combinations"](/v1/search-criteria#combination).

---

### 12.2.1 Data Types

#### Boolean (or logical data type)

A value that is true, false or unknown, so filtered for, against or ignored.

#### DateTime

The date and time

#### List

A list of options that may either be selected individually or in combination.

#### ListNumeric

A list of options that equate to a particular numerical value.

#### Numeric

Fixed precision and scale numbers, functionally equivalent to decimal.

#### Textual

A single character or string.

#### Time

The Time and no date.

### 12.2.2 Condition Types

#### Any

Enables the selection of a combination of values when used in conjunction with a list.

#### GreaterThan

All values greater than the chosen.

#### GreaterThanEqual

All values greater than or equal to the chosen.

#### LessThan

All values less than the chosen.

#### LessThanEqual

All values less than or equal to the chosen.

#### Equal

Only values equal to the chosen, in the case of a list presenting the user with a dropdown rather than a series of selection options.

#### NotEqual

All values that do not equal the chosen.

#### Contains

Includes the specified text.

#### NotContains

Does not include the specified text.

#### Range

All values that exist within the chosen range.

#### StartsWith

Values begin with specified characters.

#### EndsWith

Values end with specified characters.

### 12.2.3 Data Types and Conditions Combinations

The table of supported **Conditions** for each **Data Type**.

| Condition / Data Type | Boolean | DateTime | List | ListNumeric | Numeric | Textual | Time |
|---|---|---|---|---|---|---|---|
| **Any** | | | | | | | |
| **GreaterThan** | | | | | | | |
| **GreaterThanEqual** | | | | | | | |
| **LessThan** | | | | | | | |
| **LessThanEqual** | | | | | | | |
| **Equal** | | | | | | | |
| **NotEqual** | | | | | | | |
| **Contains** | | | | | | | |
| **NotContains** | | | | | | | |
| **Range** | | | | | | | |
| **StartsWith** | | | | | | | |
| **EndsWith** | | | | | | | |

CHAPTER 13

# Media File Metadata

The needed metadata to get *User (Agent)* recorded media files from *Recorder*.

## 13.1 Required Media File Metadata

The currently supported and required metadata by *Coach* for getting media files from *Recorder* for *User (Agent)*.

### 13.1.1 RecordingID

The database field name for recorder media file identifier or id.

### 13.1.2 RecordingFileName

The database field name for the recorder media file name.

### 13.1.3 RecorderUserID

The database field name for the recorder media file belonging user (agent).

### 13.1.4 Date

The database field name for the recorder media file date time stamp.

---

**Note:**

When the particular *Recorder* is acquired by *REST API*, for `mediaFileMetadataCollection` will be sent as collection of this four instances of `MediaPlayerMetadata`.
Then there is only need for setting matching [Recorder's](/v1/recorder) `fieldName` and its [Database Data Type](/v1/media-metadata#database-data-types).

---

## 13.2 Media File Metadata Model

Represents the **Media File Metadata** model as value object with available properties and its behaviors used for *Recorder*.

| Name | Description | Type | Required | Read-only |
|---|---|---|---|---|
| `fieldName` | The database field name to match the meaning of `localFieldName`. | `string(64)` | yes | no |
| `databaseDataType` | The [database data type](/v1/media-metadata#database-data-types) for the db `fieldName`. | `DatabaseDataType` | yes | no |
| `localFieldName` | *Coach* name for `fieldName`. | `string` | N/A | yes |
| `localDatabaseDataType` | The *Coach* [database data type](/v1/media-metadata#database-data-types) for the db `fieldName`. | `DatabaseDataType` | N/A | yes |

**Note:** Note that lookup values for setting `databaseDataType` are part of [Recorder Domain Model](/v1/recorder#recorder-model).

## 13.3 Database Data Type Model

Represents the supported *Database Data Type* as value object.

| Name | Description | Type | Required | Read-only | Default |
|---|---|---|---|---|---|
| `id` | The `name` as uppercase version. | `string` | N/A | yes | |
| `name` | The database data type name. | `string` | N/A | yes | |
| `length` | The database data type length, applicable only to string/char values. | `integer` | no | no | `0` |

**Note:** Note that only the length can be changed the `id` and `name` are fixed values!

### 13.3.1 Database Data Types

The supported database data types are:

- **Integer**
- **NVarchar** (overrides default length to 255)
- **Varchar** (overrides default length to 255)
- **Bit**
- **Datetime**

CHAPTER 14

# Recorder Media Player

**Recorder Media Player** is combination of *Recorder* and *Media Player* that is used for *User (Agent)* for getting recorded media file from *Recorder* and playing it in way that is described in *Media Player*.

## 14.1 Recorder Media Player Domain Model

Represent the **Recorder Media Player** domain model with available properties and its behaviors.

**Note:** Note that domain model is used for write methods *POST* (*Create*) and *PUT* (*Update*) and as result of read-only method *GET/:id* (*GetById*).

| Name | Description | Type | Required | Read-only | Default |
|---|---|---|---|---|---|
| `id` | Representing **Recorder Media Player** identifier. | `guid` | yes | yes | |
| `name` | The name of **Recorder Media Player**. | `string(255)` | yes | no | |
| `recorderId` | The *Recorder* reference. | `guid` | yes | no | |
| `mediaPlayerId` | The *Media Player* reference. | `guid` | yes | no | |
| `isActive` | Denotes whether the **Recorder Media Player** is active. | `bool` | yes | no | active (`true`) |
| **Lookups** | | | | | |
| `recorders` | The lookup dictionary of *Recorder*. Used to set the `redorderId`. | `dictionary(guid, string)` | N/A | N/A | N/A |
| `mediaPlayers` | The lookup dictionary of *Media Players*. Used to set the `mediaPlayerId`. | `dictionary(guid, string)` | N/A | N/A | N/A |

**Note:** The **Recorder Media Player** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Recorder Media Player** properties are capitalized (eg. `Id`, `Name`,..)!

**Danger:**
Note that only one unique combination of *Recorder* and *Media Player* can be set while creating/updating **Recorder Media Player**.
Matching same *Recorder* and *Media Player* twice will result as exception from server.

## 14.2 Recorder Media Player List Model

Represent the **Recorder Media Player** list model with available properties.

**Note:**

The list model used only to list **Media Players** with *GET* (*GetAll*) method.
Note that list model can change by adding/removing properties depending what users of *Coach REST API* will need in future.

| Name | Description | Type |
|---|---|---|
| `id` | Representing **Recorder Media Player** identifier. | `guid` |
| `name` | The name of **Recorder Media Player**. | `string` |
| `isActive` | Denotes whether the **Recorder Media Player** is active. | `string` |
| `recorderName` | The name of *Recorder*. | `string` |
| `mediaPlayerName` | The name of *Media Player*. | `string` |

**Note:** The **Recorder Media Player** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Recorder Media Player** properties are capitalized (eg. `Id`, `Name`,..)!

## 14.3 List of Recorder Media Players

The list of **Recorder Media Players** for *Current Tenant* .

### 14.3.1 Default REST approach

GET /api/v1/:tenantCode/recorder-players

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` array of [Recorder Media Player List Model](/v1/recorder-player#recorder-player-list-model).
- If there is an error: `JSON` *Client Errors* object.

### 14.3.2 C# Wrapper approach

```
RecorderMediaPlayerWrapper(int tenantCode, string apiKey, string apiSecret).GetAll();
```

### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Current Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Current Tenant* API Secret provided by **Qualtrak**.

### Return value

- If there is no error: `ResaultContent<ICollection<RecorderMediaPlayer>>.Result` object collection of the [Recorder Media Player List Model](/v1/recorder-player#recorder-player-list-model).
- If there is an error: `ResaultContent<ICollection<RecorderMediaPlayer>>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<RecorderMediaPlayer, RecorderMediaPlayerList> recorderMediaPlayerWrapper = new Recor
ResponseContent<ICollection<RecorderMediaPlayerList>> response = recorderMediaPlayerWrapper.GetAll();

if (response.Result != null)
{
     // Use Result as List of recorder Media Players for displaying.
     ICollection<RecorderMediaPlayerList> recorderMediaPlayers = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 14.4 Get Recorder Media Player by Id

The **Recorder Media Player** by requested Id for *current Tenant*.

### 14.4.1 Default REST approach

GET /api/v1/:tenantCode/recorder-players/:id

### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Recorder Media Player** id, a valid and non-empty `guid`.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` as the [Recorder Media Player Domain Model](/v1/recorder-player#recorder-player-model) object.
- If there is an error: `JSON` as the *Client Errors* object.

### 14.4.2 C# Wrapper approach

```
RecorderMediaPlayerWrapper(int tenantCode, string apiKey, string apiSecret).GetById(Guid id);
```

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Current Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Current Tenant* API Secret provided by **Qualtrak**.
- `id` The **Recorder Media Player** id, a valid and non-empty `guid`.

#### Return value

- If there is no error: `ResaultContent<RecorderMediaPlayer>.Result` object as the [Recorder Media Player Domain Model](/v1/recorder-player#recorder-player-model).
- If there is an error: `ResaultContent<RecorderMediaPlayer>.Error` object. See more in *Client Errors*

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid recorderMediaPlayerId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<RecorderMediaPlayer, RecorderMediaPlayerList> recorderMediaPlayerWrapper = new Recor
ResponseContent<RecorderMediaPlayer> response = recorderMediaPlayerWrapper.GetById(recorderMediaPlaye

if (response.Result != null)
{
     // Use Result as requested Recorder Media Player for displaying.
     RecorderMediaPlayer recorderMediaPlayer = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 14.5 Create Recorder Media Player

The creation of new **Recorder Media Player** for *current Tenant*.

### 14.5.1 Default REST approach

POST /api/v1/:tenantCode/recorder-players

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `recorderMediaPlayer JSON` representation of **Recorder Media Player properties** sent via *Request HTTP Header*.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` representation of newly created **Recorder Media Player** as the [Recorder Media Player Domain Model](/v1/recorder-player#recorder-player-model).
- If there is an error: `JSON` *Client Errors* object.

### 14.5.2 C# Wrapper approach

```
RecorderMediaPlayerWrapper(int tenantCode, string apiKey, string apiSecret).Create(RecorderMediaPlaye
```

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Current Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Current Tenant* API Secret provided by **Qualtrak**.
- `recorderMediaPlayer` The **Recorder Media Player** model constructed from **Recorder Media Player properties**.

#### Return value

- If there is no error: `ResaultContent<RecorderMediaPlayer>.Result` object as the [Recorder Media Player Domain Model](/v1/recorder-player#recorder-player-model).
- If there is an error: `ResaultContent<recorderMediaPlayer>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<RecorderMediaPlayer, RecorderMediaPlayerList> recorderMediaPlayerWrapper = new Recorc
// Get default data and lookup for recorder media players
```

```
RecorderMediaPlayer newRecorderMediaPlayer = recorderMediaPlayerWrapper.GetById(new Guid()).Result;
newRecorderMediaPlayer.Name = "Recorder Media Player created from test";

// Set recorder and mediaPlayer from lookups key.
newRecorderMediaPlayer.RecorderId = newRecorderMediaPlayer.Recorders.FirstOrDefault().Key;
newRecorderMediaPlayer.MediaPlayerId = newRecorderMediaPlayer.MediaPlayers.FirstOrDefault().Key;

ResponseContent<RecorderMediaPlayer> response = recorderMediaPlayerWrapper.Create(newRecorderMediaPla

if (response.Result != null)
{
     // Use Result as newly created Recorder Media Player for display.
     RecorderMediaPlayer recorderMediaPlayer = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 14.6 Update Recorder Media Player

Updates already existent **Recorder Media Player** for *current Tenant* .

### 14.6.1 Default REST approach

PUT /api/v1/:tenantCode/recorder-players/:id

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Recorder Media Player** id, a valid and non-empty `guid`.
- `recorderMediaPlayer` `JSON` representation of **recorder Media Player properties** sent via *Request HTTP Header*.

**Danger:**
Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.
If you don't want to have in Web Server turned on the `PUT` verb method read more in *Getting Started*.

#### Return value

- If there is no error: `JSON` representation of uodated **Recorder Media Player** as the [Recorder Media Player Domain Model](/v1/recorder-player#recorder-player-model) object.
- If there is an error: `JSON` *Client Errors* object.

### 14.6.2 C# Wrapper approach

```
RecorderMediaPlayerWrapper(int tenantCode, string apiKey, string apiSecret).Update(RecorderMediaPlaye
```

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Current Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Current Tenant* API Secret provided by **Qualtrak**.
- `recorderMediaPlayer` The **Recorder Media Player** model constructed from **Recorder Media Player properties** and `Id` must be provided in it. If not `ArgumentException` will be thrown!
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `PUT` method. Default is `false` or use `PUT` method!

#### Return value

- If there is no error: `ResaultContent<RecorderMediaPlayer>.Result` object as the [Recorder Media Player Domain Model](/v1/recorder-player#recorder-player-model).
- If there is an error: `ResaultContent<RecorderMediaPlayer>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid recorderMediaPlayerId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<RecorderMediaPlayer, RecorderMediaPlayerList> recorderMediaPlayerWrapper = new Recorc
RecorderMediaPlayer recorderMediaPlayer = recorderMediaPlayerWrapper.GetById(recorderMediaPlayerId).R
recorderMediaPlayer.Name = "Recorder Media Player updated from test";
recorderMediaPlayer.RecorderId = recorderMediaPlayer.Recorders.LastOrDefault().Key;
recorderMediaPlayer.MediaPlayerId = recorderMediaPlayer.MediaPlayers.LastOrDefault().Key;

// Update via PUT method (default).
ResponseContent<RecorderMediaPlayer> response = recorderMediaPlayerWrapper.Update(recorderMediaPlayer

// Update via POST method (use true argument).
// ResponseContent<RecorderMediaPlayer> response = recorderMediaPlayerWrapper.Update(recorderMediaPla

if (response.Result != null)
{
     // Use Result of updated Recorder Media Player for display.
     RecorderMediaPlayer updatedRecorderMediaPlayer = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 14.7 Delete Recorder Media Player

Deletes existent **Recorder Media Player** for *current Tenant* .

> **Warning:** Note that **Recorder Media Player** will not be deleted if there are any references of **Recorder Media Player** in [User](/v1/user).

### 14.7.1 Default REST approach

DELETE /api/v1/:tenantCode/recorder-players/:id

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Recorder Media Player** id, a valid and non-empty `guid`.

> **Danger:**
> Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.
> If you don't want to have in Web Server turned on the `DELETE` verb method read more in *Getting Started*.

#### Return value

- There is no return value except if there is an error, the `JSON` *Client Errors* object.

### 14.7.2 C# Wrapper approach

```
RecorderMediaPlayerWrapper(int tenantCode, string apiKey, string apiSecret).Delete(Guid id, bool upda
```

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Current Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Current Tenant* API Secret provided by **Qualtrak**.
- `id` The **Recorder Media Player** id, a valid and non-empty `guid`.
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `DELETE` method. Default is `false` or use `DELETE` method!

#### Return value

- If there is no error: no return value or `void`.
- If there is an error: `ResaultContent<RecorderMediaPlayer>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid recorderMediaPlayerId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<RecorderMediaPlayer, RecorderMediaPlayerList> recorderMediaPlayerWrapper = new Recor
// Delete via DELETE method (default).
ResponseContent response = recorderMediaPlayerWrapper.Delete(recorderMediaPlayerId);

// Delete via POST method (use true argument).
// ResponseContent response = recorderMediaPlayerWrapper.Delete(recorderMediaPlayerId, true);

if (response.Error != null)
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

CHAPTER 15

# Schedule

**Schedule** is automatically collects media for *Managers* to evaluate in the **Schedule** section of *Coach QM*. **Schedule** essential parts are [Period Types and Occurrences](/v1/period-types) that describe the interval and type of occurrence.

## 15.1 Schedule Domain Model

Represent the **Schedule** domain model with available properties and its behaviors.

**Note:** Note that domain model is used for write methods *POST* (*Create*) and *PUT* (*Update*) and as result of read-only method *GET/:id* (*GetById*).

| Name | Description | Type | Required | Read-only | Default |
|---|---|---|---|---|---|
| `id` | Representing **Schedule** identifier. | `guid` | yes | yes | |
| **Main Details** | | | | | |
| `name` | The name of **Schedule**. | `string(500)` | yes | no | |
| `description` | The description of **Schedule**. | `string(1024)` | no | no | |
| `startDate` | The start date that triggers **Schedule**. It can be today or some date in future. | `datetime` | yes | no | |
| `maxCallPerAgent` | The max number of media calls per *Agent*. Valid range is from 1 to 99 | `short` | yes | no | `5` |
| `isActive` | Denotes whether the **Schedule** state is active or inactive. | `boolean` | yes | no | active (`false`) |
| `isArchived` | Denotes whether the **Schedule** state is archived or not. | `boolean` | yes | no | not archived (`false`) |
| `createdAt` | The **Schedule** creation date. | `datetime` | yes | yes | |
| `updatedAt` | The date when **Schedule** was last time updated. | `datetime` | yes | yes | |
| `lastRunAt` | The date when **Schedule** was last time run by *Scheduler* engine. | `datetime` | no | yes | `null` |
| **Period & Occurrence** | | | | | |
| `periodType` | The **Schedule** [Period Types](/v1/period-types). | `byte` | yes | no | Daily (`0`) |
| `occurrenceType` | The **Schedule** [Occurrences Types](/v1/period-types). | `byte` | yes | no | Infinite (`0`) |
| `occurrenceTimesNumber` | The number of times that **Schedule** will [occur](/v1/period-types). | `short` | no | no | `0` |
| `endDate` | The **Schedule** end date for [End Date Occurrence](/v1/period-types#until-end-date). | `datetime` | no | no | `null` |
| `customDateFrom` | The **Schedule** [Custom Period date from](/v1/period-types#custom). | `datetime` | no | no | `null` |
| `customDateTo` | The **Schedule** [Custom Period date to](/v1/period-types#custom). | `datetime` | no | no | `null` |
| **Levels & Search Criteria** | | | | | |
| `levels` | The **Schedule** levels, see more in [Tree Item Types](/v1/tree-item-type#levels-and-item-types). | `array(guid)` | no | no | |
| `searchCriteria` | The **Schedule** collection of [Search Criteria](/v1/search-criteria). | `array(SearchCriteria)` | no | no | |
| **Lookups** | | | | | |
| `periodTypes` | The dictionary of supported [Period Types](/v1/period-types#period-types).\| `dictionary(byte, string)` | N/A | N/A | N/A \| | |
| `occurrenceTypes` | The dictionary of supported [Occurrences](/v1/period-types#occurrences).\| `dictionary(byte, string)` | N/A | N/A | N/A \| | |
| `searchDataTypesCollection` | The collection of [Data Types and its supported Conditions](/v1/search-criteria#data-condition-model) | `array(SearchDataTypeAndCondition)` | N/A | N/A | N/A |

**Note:** The **Schedule** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Schedule** properties are capitalized (eg. `Id`, `Name`,..)! \

**Note:** Note that *Required* properties are changed depending on wanted Period Type and Occurrence. See more in whole list of [required properties for Period Types and Occurrences](/v1/period-types#period-occurrence-required).

**Danger:**
Beware that lookup for `levels` is not part of **Schedule** domain model. [Unit, Team or Agent tree item types](/v1/tree#levels-and-item-types) are valid items of [Tenant Tree](/v1/tree) to be assigned as **Schedule** `levels`.

To assign values to `levels` you need to call [GetTree method](/v1/tree#tenant-tree-item) and use [Tenant Tree Item Model](/v1/tree#tenant-tree-item-model) `id` as a individual `guid` for `levels`.

**Warning:** Note that if no `levels` are assigned then the **Schedule** can not be active. So even the `isActive` is set to `true` and there are no `levels` assigned the `isActive` will be set to `false`.

## 15.2 Schedule List Model

Represent the **Schedule** list model with available properties.

**Note:**

The list model used only to list **Schedules** with *GET* (*GetAll*) method.

Note that list model can change by adding/removing properties depending what schedules of *Coach REST API* will need in future.

| Name | Description | Type |
|---|---|---|
| `id` | Representing **Schedule** identifier. | `guid` |
| `name` | The name of **Schedule**. | `string` |
| `description` | The **Schedule** description. | `string` |
| `startDate` | The **Schedule** start date. | `datetime` |
| `isActive` | Denotes whether the **Schedule** state is active or inactive. | `boolean` |
| `isArchived` | Denotes whether the **Schedule** state is archived or not. | `boolean` |
| `periodType` | The **Schedule** [Period Type](/v1/period-types#period-types). | `string` |
| `occurrenceType` | The **Schedule** [Occurrence Type](/v1/period-types#occurrences). | `string` |
| `maxCallsPerAgent` | The max number of media calls per *Agent*. | `short` |
| `nextRunAt` | The **Schedule** next run at data as relative time eg. “in 2 days”. | `string` |
| `lastRunAt` | The **Schedule** last run at data as relative time eg. “2 days ago”. | `string` |
| `createdSince` | The **Schedule** creation data as relative time eg. “2 days ago”. | `string` |
| `updatedSince` | The **Schedule** data of last update as relative time eg. “2 days ago”. | `string` |
| `levels` | The collection of [Schedule Levels List model](/v1/schedule#schedule-level-list-model). | `array(ScheduleLevelList)` |

**Note:** The **Schedule** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Schedule** properties are capitalized (eg. `Id`, `Name`,..)!

## 15.3 Schedule Level List Model

Represent the **Schedule** Level List model with available properties. Used as collection in [Schedule List model](/v1/schedule#schedule-list-model).

| Name | Description | Type |
|---|---|---|
| `name` | The *Unit*, *Team* or *Agent* name. | `string` |
| `tenantTreeLevelType` | The [Unit, Team or Agent tree item type](/v1/tree#levels-and-item-types). | `string` |

## 15.4 List of Schedules

The list of **Schedules** for *Tenant*.

### 15.4.1 Default REST approach

GET /api/v1/:tenantCode/schedules

#### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` array of [Schedule List Model](/v1/schedule#schedule-list-model).
- If there is an error: `JSON` *Client Errors* object.

### 15.4.2 C# Wrapper approach

```
ScheduleWrapper(int tenantCode, string apiKey, string apiSecret).GetAll();
```

#### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Tenant* API Secret provided by **Qualtrak**.

### Return value

- If there is no error: `ResaultContent<ICollection<Schedule>>.Result` object as collection of the [Schedule List Model](/v1/schedule#schedule-list-model).
- If there is an error: `ResaultContent<ICollection<Schedule>>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

ITreeApiWrapper<Schedule, ScheduleList> scheduleWrapper = new ScheduleWrapper(tenantCode, key, secret
ResponseContent<ICollection<ScheduleList>> response = scheduleWrapper.GetAll();

if (response.Result != null)
{
     // Use Result as List of Schedules for displaying.
     ICollection<ScheduleList> schedules = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 15.5 Get Schedule by Id

The **Schedule** by requested Id for *Tenant*.

### 15.5.1 Default REST approach

GET /api/v1/:tenantCode/schedules/:id

### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Schedule** id, a valid and non-empty `guid`.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

### Return value

- If there is no error: `JSON` as the [Schedule Domain Model](/v1/schedule#schedule-model) object.
- If there is an error: `JSON` as the *Client Errors* object.

### 15.5.2 C# Wrapper approach

```
ScheduleWrapper(int tenantCode, string apiKey, string apiSecret).GetById(Guid id);
```

#### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Tenant* API Secret provided by **Qualtrak**.
- `id` The **Schedule** id, a valid and non-empty `guid`.

#### Return value

- If there is no error: `ResaultContent<Schedule>.Result` object as the [Schedule Domain Model](/v1/schedule#schedule-model).
- If there is an error: `ResaultContent<Schedule>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid scheduleId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<Schedule, ScheduleList> scheduleWrapper = new ScheduleWrapper(tenantCode, key, secret
ResponseContent<Schedule> response = scheduleWrapper.GetById(scheduleId);

if (response.Result != null)
{
     // Use Result as requested Schedule for displaying.
     Schedule schedule = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 15.6 Create Schedule

The creation of new **Schedule** for *Tenant*.

> **Warning:** To assign `levels` you'll need to get [Tenant Tree](/v1/tree) and use [Unit, Team or Agent](/v1/tree#levels-and-item-types) items as Levels!

### 15.6.1 Default REST approach

POST /api/v1/:tenantCode/schedules

#### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.
- `schedule` `JSON` representation of [Schedule Domain Model](/v1/schedule#schedule-model) sent via *Request HTTP Header*.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` representation of newly created **Schedule** as the [Schedule Domain Model](/v1/schedule#schedule-model).
- If there is an error: `JSON` *Client Errors* object.

### 15.6.2 C# Wrapper approach

```
ScheduleWrapper(int tenantCode, string apiKey, string apiSecret).Create(Schedule schedule);
```

#### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Tenant* API Secret provided by **Qualtrak**.
- `schedule` The **Schedule** model constructed from **Schedule properties**.

#### Return value

- If there is no error: `ResaultContent<Schedule>.Result` object as the [Schedule Domain Model](/v1/schedule#schedule-model).
- If there is an error: `ResaultContent<Schedule>.Error` object. See more in *Client Errors*

#### Example usage

```
ICollection<Guid> levels = new List<Guid>();
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

TreeWrapper treeWrapper = new TreeWrapper(tenantCode, key, secret);
ResponseContent<TenantTreeItem> responseTree = treeWrapper.GetTree();

```

```
if (responseTree.Result != null)
{
    levels.Add(responseTree.Result.Items.Where(x => x.TreeItemType == "Unit").Select(x => x.Id).Sing
}

ITreeApiWrapper<Schedule, ScheduleList> scheduleWrapper = new ScheduleWrapper(tenantCode, key, secret
// Get default data and lookup for schedules
Schedule newSchedule = scheduleWrapper.GetById(new Guid()).Result;
newSchedule.Name = "Tester";
newSchedule.StartDate = DateTime.Now;
newSchedule.Levels = levels;

ResponseContent<Schedule> response = scheduleWrapper.Create(newSchedule);

if (response.Result != null)
{
     // Use Result as newly created Schedule for display.
     Schedule schedule = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 15.7 Update Schedule

Updates already existent **Schedule** for *Tenant*.

> **Warning:** To assign `levels` you'll need to get [Tenant Tree](/v1/tree) and use [Unit, Team or Agent](/v1/tree#levels-and-item-types) items as Levels!

### 15.7.1 Default REST approach

PUT /api/v1/:tenantCode/schedules/:id

#### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Schedule** id, a valid and non-empty `guid`.
- `schedule` `JSON` representation of [Schedule Domain Model](/v1/schedule#schedule-model) sent via *Request HTTP Header*.

> **Danger:**
> Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*
> If you don't want to have in Web Server turned on the `PUT` verb method read more in *Getting Started*.

#### Return value

- If there is no error: `JSON` representation of uodated **Schedule** as the [Schedule Domain Model](/v1/schedule#schedule-model).
- If there is an error: `JSON` *Client Errors* object.

### 15.7.2 C# Wrapper approach

```
ScheduleWrapper(int tenantCode, string apiKey, string apiSecret).Update(Schedule schedule, bool updat
```

#### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Tenant* API Secret provided by **Qualtrak**.
- `schedule` The **Schedule** model constructed from **Schedule properties** and `Id` must be provided in it. If not `ArgumentException` will be thrown!
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `PUT` method. Default is `false` or use `PUT` method!

#### Return value

- If there is no error: `ResaultContent<Schedule>.Result` object as the [Schedule Domain Model](/v1/schedule#schedule-model).
- If there is an error: `ResaultContent<Schedule>.Error` object. See more in *Client Errors*.

#### Example usage

```
ICollection<Guid> levels = new List<Guid>();
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid scheduleId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

TreeWrapper treeWrapper = new TreeWrapper(tenantCode, key, secret);
ResponseContent<TenantTreeItem> responseTree = treeWrapper.GetTree();

if (responseTree.Result != null)
{
    levels.Add(responseTree.Result.Items.Where(x => x.TreeItemType == "Unit").Select(x => x.Id).LastO
}

ITreeApiWrapper<Schedule, ScheduleList> scheduleWrapper = new ScheduleWrapper(tenantCode, key, secret
Schedule schedule = scheduleWrapper.GetById(scheduleId).Result;
schedule.Name = "Test Schedule Updated";
schedule.Levels = levels;

// Update via PUT method (default).
ResponseContent<Schedule> response = scheduleWrapper.Update(schedule);
```

```

// Update via POST method (use true argument).
// ResponseContent<Schedule> response = scheduleWrapper.Update(schedule, true);

if (response.Result != null)
{
     // Use Result of updated Schedule for display.
     Schedule updatedSchedule = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

## 15.8 Delete Schedule

Deletes existent **Schedule** for *Tenant*.

> **Warning:**
> Note that **Schedule** will not be deleted but rather flagged as `isArchived` if it was run at least once by *Schedule* engine and there are associated recordings (media) with this **Schedule**.
> If **Schedule** has never run and there are no associated recordings (media) to it, it will be deleted permanently with all assigned **Schedule** levels.

### 15.8.1 Default REST approach

DELETE /api/v1/:tenantCode/schedules/:id

#### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.
- `id` The **Schedule** id, a valid and non-empty `guid`.

> **Danger:**
> Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.
> If you don't want to have in Web Server turned on the `DELETE` verb method read more in *Getting Started*

#### Return value

- There is no return value except if there is an error, the `JSON` *Client Errors* object.

### 15.8.2 C# Wrapper approach

```
ScheduleWrapper(int tenantCode, string apiKey, string apiSecret).Delete(Guid id, bool updateViaPost =
```

### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Tenant* API Secret provided by **Qualtrak**.
- `id` The **Schedule** id, a valid and non-empty `guid`.
- `updateViaPost` Set to `true` if in your Web Server you don't want to enable `DELETE` method. Default is `false` or use `DELETE` method!

### Return value

- If there is no error: no return value or `void`.
- If there is an error: `ResaultContent<Schedule>.Error` object. See more in *Client Errors*

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";
Guid scheduleId = new Guid("f4fe3ea7-ed2a-41dd-acd2-91c45c8b4891");

ITreeApiWrapper<Schedule, ScheduleList> scheduleWrapper = new ScheduleWrapper(tenantCode, key, secret
// Delete via DELETE method (default).
ResponseContent response = scheduleWrapper.Delete(scheduleId);

// DELETE via POST method (use true argument)..
// ResponseContent response = scheduleWrapper.Delete(scheduleId, true);

if (response.Error != null)
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

CHAPTER 16

# Period Types and Occurrences

## 16.1 Period Types

The **Period Types** are essential part of *Schedule* and currently these **Period Types** are supported:

### 16.1.1 Daily

The *Schedule* will run every day.

### 16.1.2 Weekly

The *Schedule* will run every week.

### 16.1.3 Monthly

The *Schedule* will run every month.

### 16.1.4 Quarterly

The *Schedule* will run quarterly or every three months.

### 16.1.5 Custom

A custom form and to date for a one-off search. The `customDateFrom` and `customDateTo` dates must be set to cover a period prior to the *Schedule's* `startDate`.

## 16.2 Occurrences

The **Occurrences** are part of **Period Types** describing the type of occurrence on specific **Period Type**. **Occurrences** are triggered by the *Schedule's* `startDate`.

Currently supported **Occurrences** are:

### 16.2.1 None

Used only for *Custom* **Period Type** since it will always have only one occurrence.

### 16.2.2 Infinite

Will occur infinitely.

### 16.2.3 Number Of Times

Will occur for the specified number of times described through number declared *Schedule's* `occurrenceTimesNumber` property.

### 16.2.4 Until End Date

Will occur until some specified *Schedule's* `endDate` that is higher or equal of `startDate`.

## 16.3 Period Types and Occurrences Combinations

The combination of **Period Types** and the supported **Occurrences** for **Period Type**.

| Period Occurrence | None | Infinite | Number Of Times | Until End Date |
|---|---|---|---|---|
| **Daily** | | | | |
| **Weekly** | | | | |
| **Monthly** | | | | |
| **Quarterly** | | | | |
| **Custom** | | | | |

## 16.4 Period Types and Occurrences with Required [Schedule](/v1/schedule) Model Properties

The table of **Occurrences** with valid **Period Types** and required *Schedule* model properties. Check mark denotes required *Schedule* model property for **Period Type Occurrence**.

| Occurrences | Period Type(s) | `occurrenceTimesNumber` | `endDate` | `customDateFrom` | `customDateTo` |
|---|---|---|---|---|---|
| | | *Schedule* Model Properties | | | |
| **None** | **Custom** | | | | |
| **Infinite** | **Daily**, **Weekly**, **Monthly** and **Quarterly** | | | | |
| **Number Of Times** | **Daily**, **Weekly**, **Monthly** and **Quarterly** | | | | |
| **Until End Date** | **Daily**, **Weekly**, **Monthly** and **Quarterly** | | | | |

**Danger:** Beware that the *API* will raise an exception when not needed property is sent for some combination of **Period Type** and **Occurrence**. Eg. if for *Daily* **Period Type** as *Infinite* **Occurrence** is sent not needed `endDate` property, this will raise an exception with message: *"Daily period type of Infinite occurrence type cannot have an end date assigned to it!"*.

CHAPTER 17

# License

To use *Coach QM*, there is a need to obtain valid license from *Coach Licensing Server*.

The **License** can be:

**Trial**

The trial **License** has expiry date until is valid, and after its expired, the access to *Coach QM* is denied for all users, but *Coach Console* is still accessible and fully functional.

**Forever**

The **License** last forever and it has count of bought **Licenses** that is subtracted for every creation of an active agent. When the count is exceeded, there is no way to make active agents, except of buying more licenses.

**Note:** Note that the information of Expiry Date of **License**, Available and Used **Licenses** you can get through *Tenant* properties.

## 17.1 Obtain License

Obtains **License** for *Current Tenant*.

**Danger:** Beware that *Tenant* properties `customerId` and `customerCode` needs to be updated with values given by *Qualtrak* received upon **License** purchase or on request for trial **License**.

### 17.1.1 Default REST approach

GET /api/v1/:tenantCode/license

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.

**Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in [Getting Started](/v1).

#### Return value

- There is no return value except if there is an error, the `JSON` *Client Errors* object.

### 17.1.2 C# Wrapper approach

```
LicenseWrapper(int tenantCode, string apiKey, string apiSecret).Obtain();
```

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Current Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Current Tenant* API Secret provided by **Qualtrak**.

#### Return value

- If there is no error: no return value or `void`.
- If there is an error: `ResaultContent<ICollection<Error>>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

LicenseWrapper licenseWrapper = new LicenseWrapper(tenantCode, key, secret);
ResponseContent response = licenseWrapper.Obtain();

if (response.Error != null)
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

CHAPTER 18

# Tenant Tree

A **Tenant Tree** is a flattened representation of the whole hierachy including *Tenant*, recursive *Units* and its *Managers* (*User*), *Unit Tenant* and its *Managers* and *Agents* which are essentially (*Users*).

## 18.1 Graphical Tenant Tree Representation

..tree:

```
+ Tenant
|
|---+ Unit (recursive)
    | Unit manager(s)
    |
    |---+ Team
        | Team manager(s)
        | Agent(s)
```

## 18.2 Tenant Tree Item Model

Represent the **Tenant Tree** model as flattened hierarchy with read-only properties.

**Note:** Note that model is used as result of read-only method *GET* (*GetTree*).

| Name | Description | Type |
|---|---|---|
| `id` | Representing **Tenan Tree Item** identifier. | `guid` |
| **Common properties** | | |
| `treeItemId` | The id of *Tenant*, *Unit*, *Team* or *User*. | `guid` |
| `treeItemType` | The item type. *Tenant*, *Unit*, *Team* or *User*. | `string` |
| `name` | The name of *Tenant*, *Unit*, *Team* or *User*. | `string` |
| `isActive` | Denotes whether the **Tenant Tree Item** state is active or inactive. | `boolean` |
| `isDeleted` | Denotes whether the **Tenant Tree Item** state is deleted or not. | `boolean` |
| **User specific** | | |
| `recorderUserId` | The *User* identification of media files within the Recorder. | `string` |
| `recorderUserId` | The *User* identification of media files within the Recorder. | `string` |
| `haveRecorder` | Denotes whether the [Recorder](/v1/recorder) is assigned to *User*. | `boolean` |
| **Manager / Agent specific** | | |
| `managerOrAgentId` | Representing the true identifier of managership/membership not only the `userId`. | `guid` |
| **Tenant specific** | | |
| `tenantId` | Sets to each **Tenant Tree Item** `tenantId` to denote which *Tenant* it belongs. | `guid` |

**Note:** The **Tenant Tree** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Tenant Tree** properties are capitalized (eg. `Id`, `Name`,..)!

### 18.2.1 Tenant Tree Item Types

Currently supported [Tenant Tree](/v1/tree) item types are:

- Tenant
- Unit
- Unit Manager
- Team
- Team Manager
- Agent

#### Levels and Tenant Tree Item Types

[Schedule](/v1/schedule) levels are essentially a **Tenant Tree** items. But only tree items that are of type *Unit*, *Team* or *Agent* are valid as *Schedule* `levels`, other tree item types are invalid!

## 18.3 Tenant Tree Item

The root [Te:ref:*tenant-label* tree item with recursive items.

### 18.3.1 Default REST approach

GET /api/v1/:tenantCode/tree

#### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.

> **Danger:** Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` [Tenant Tree Item Model](/v1/tree#tenant-tree-item-model).
- If there is an error: `JSON` *Client Errors* object.

### 18.3.2 C# Wrapper approach

```
TreeWrapper(int tenantCode, string apiKey, string apiSecret).GetTree();
```

#### Parameters

- `tenantCode` *Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Tenant* API Secret provided by **Qualtrak**.

#### Return value

- If there is no error: `ResaultContent<TenantTreeItem>.Result` object as [Tenant Tree Item Model](/v1/tree#tenant-tree-item-model).
- If there is an error: `ResaultContent<TenantTreeItem>.Error` object. See more in *Client Errors*.

#### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

TreeWrapper treeWrapper = new TreeWrapper(tenantCode, key, secret);
ResponseContent<TenantTreeItem> response = treeWrapper.GetTree();

if (response.Result != null)
{
     // Use Result as root Tenant Tree Item.
     TenantTreeItem rootTenantItem = response.Result;

     // Root Units can be invoked
     ICollection<TenantTreeItem> rootUnits = rootTenantItem.Items;

     // .... use recursion to get full hierarchy for displaying
}
else
```

```
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```

CHAPTER 19

# Recording Evaluations

Returns collection of *Evaluations* for given *Recording* call Id.

## 19.1 Recording Evaluations List Model

Represent the **Recording Evaluations** list model with available properties.

**Note:**

The list model used only to list **Recording Evaluations** with *GET* (*GetAll*) method.

Note that list model can change by adding/removing properties depending what users of *Coach REST API* will need in future.

| Name | Description | Type |
|---|---|---|
| `id` | Representing **Evaluation** identifier. | `guid` |
| `reference` | The **Evaluation** reference. | `string` |
| `score` | The **Evaluation** score. | `int (nullable)` |
| `date` | The date when the **Evaluation** took place. | `datetime` |
| `isCompleted` | Whether the **Evaluation** is completed. | `boolean` |
| `isAutoFailed` | Wheather the **Evaluation** has failed. | `boolean` |
| `comments` | The **Evaluations** comments. | `string` |

**Note:** The **Recording Evaluations** properties names (*Name* column) is for default usage by `JSON`, for `C# Wrapper` usage the **Recording Evaluations** properties are capitalized (eg. `Id`, `Name`,..)!

## 19.2 List of Evaluations for Recording Call Id

The list of **Recording Evaluations** for *current Tenant*.

### 19.2.1 Default REST approach

GET /api/v1/:tenantCode/recording-evaluations/:callId

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `callId` The **Recording** call id, a valid and non-empty `string`.

> **Danger:**
> Remember to remove `/`, `.`, `:` and `\` characters from `callId` string. If not the response will be 404 due to invalid URL.
>
> Remember to add *API Key* as *customer*key* and *API Secret* as *customer*secret* into your *Request HTTP Header*. See more in *Getting Started*.

#### Return value

- If there is no error: `JSON` array of *Recording Evaluations*.
- If there is an error: `JSON` *Client Errors* object.

### 19.2.2 C# Wrapper approach

```
RecordingEvaluationWrapper(int tenantCode, string apiKey, string apiSecret).GetEvaluationsByCallId(st
```

#### Parameters

- `tenantCode` *Current Tenant* code, a valid `integer` greater or equal to 1000.
- `apiKey` *Current Tenant* API Key provided by **Qualtrak**.
- `apiSecret` *Current Tenant* API Secret provided by **Qualtrak**.
- `callId` The **Recording** call id, a valid and non-empty `string`.

> **Danger:** Remember to remove `/`, `.`, `:` and `\` characters from `callId` string. If not the response will be 404 due to invalid URL.

#### Safe Call Id Extension Method

Create the C# `string` Extension Method to make `callId` safe and call it always to make safe `callId` to get `Evauluations`:

```
public static class StringExtensions
{
    public static string ToSafeCallId(this string callId)
    {
        string result = callId.Replace(@"\", "")
                              .Replace("/", "")
                              .Replace(".", "")
                              .Replace(":", "");

        return result;
    }
```

```
}

// and call it as:
string callId = "10.1.1.1:300/recording/2012/01/01/a.wav".ToSafeCallId();
// callId => "10111300recording20120101awav"
```

### Return value

- If there is no error: `ResaultContent<ICollection<RecordingEvaluationList>>.Result` object collection of the *Recording Evaluations*.
- If there is an error: `ResaultContent<ICollection<RecordingEvaluationList>>.Error` object. See more in *Client Errors*.

### Example usage

```
int tenantCode = 1000;
string key = "ddZXdAZvWefFqxAEH62u";
string secret = "wx6GiQggg9YRH89XT5aKoY2qZLVquYjxARtgZhuGoFQX5w6Lws";

// Preffered way of creating ``callId`` by calling the ``string`` extension method ``ToSafeCallId``.
// See implementation in "Safe Call Id Extension Method"!
string callId = "10.1.1.1:300/recording/2012/01/01/a.wav".ToSafeCallId();

// Another way but error prone!
// string callId = "/10.1.1.1:300/recording/2012/01/01/a.wav".Replace("/", "").Replace(".", "").Repla

RecordingEvaluationWrapper recordingEvaluationWrapper = new RecordingEvaluationWrapper(tenantCode, ke
ResponseContent<ICollection<RecordingEvaluationList>> response = recordingEvaluationWrapper.GetEvalua

if (response.Result != null)
{
     // Use Result as requested Recording Evaluations for displaying.
     ICollection<RecordingEvaluationList> recordingEvaluations = response.Result;
}
else
{
    // TODO: The error handling...
    Console.WriteLine(response.Error);
}
```