BoatLogger Web Service API

Document Version 1.1, Updated 05th October 2014

Contents

Maintenance History

Document Version	Date	Author	Comments
1.0	11 February 2014	Advent Phang	Original
1.1	05 October 2014	Alvin Chua	Added UserInfo API

Preamble

The BoatLogger REST Web Service API (BLWS) provides services for third-party developers to build devices that can interact with BoatLogger user resources.

Device Activation

Before a device can interact with the BLWS, it must be activated. Activation is performed over a web browser, by navigating user to:

`/Account/ActivateDevice?deviceId={device-id}&deviceType={device-type}`

Query	Type	Description
{device-id}	string	Device ID is a 32 alpha-numeric characters string that uniquely identifies a device/application. Device must not share the same device id, even though they were of the same device type. Device ID could be a generated GUID.
{device-type}	string	Optional. Device type is a descriptive text of up to 255 characters that can be used to describe the type of the device. For example: “BoatLogger Smartphone”.

On successful user activation, server will return the activation key by redirecting the browser to:

`/Account/ActivateDeviceSuccess?key={activation-key}`

Query	Type	Description
{activation-key}	string	Activation key is a 32 alpha-numeric characters string that device must use in all subsequent BLWS requests.

If a device has been activated before for the authenticated user, server will return the same activation key. In this situation, the device type is not updated.

On failed user activation, server will redirect the browser to:

`/Account/ActivateDeviceFailed?err={error-message}`

Query	Type	Description
{error-message}	string	Activation failure message.

Authenticating REST Requests

The BLWS uses the standard HTTP `Authorization` header to pass authentication information. (The name of the standard header is unfortunate because it carries authentication information, not authorisation). The `Authorization` header has the following form:

```
Authorization: BLWS key={activation-key}
```

The activation key is obtained by performing device activation. See Device Activation for more information on device activation.

Failure to include a valid activation key will cause the server to reject with `401 Unauthorized` HTTP status code.

Versioning Requests

The BLWS uses the optional `ver` field in the query string to identify the requested service version. If the version is not specified, or if the value specified is not invalid, the server fall backs the latest stable version. Specifying the version is highly recommended.

The current stable API version is **1.0**.

Sample API URL with version value in query string:

API	Description
GET /api/Event?ver=1.0&segment=2	Gets all the events that is associated to a specific log book segment.
POST /api/Event?ver=1.0	Adds a new yacht event.

API HTTP Response Code

Depending on service outcome, BLWS returns different HTTP status code. Below lists the most common response code.

HTTP Status Code	Description
200 OK	Successful API call.
201 Created	An API call that has resulted in creation of a specific resource, one that can be referenced by the URI specified in the response's Location header field. The created entity is also returned as part or whole of the content.
204 No Content	A successful API call that has no content.

400 Bad Request	Failed API call. Response content may contain the following information:
	
	`Message : string`
	`ErrorCode : ErrorCode`
	`FieldName : string`
	
	Message is a short English description on the error. ErrorCode is a numeric value for the specific BLWS error. FieldName may identifies the field name in the request packet that is causing the error.

401 Unauthorized	Missing authorization header, invalid authorization scheme, missing or invalid activation key.
404 Not Found	An entity as referenced by a specific ID is invalid.
415 Unsupported Media Type	The entity of the request is in a format that is not supported by the requested service.

Date Time and Date Time Offset Serialisation

All date time and date time offset are serialised according to the ISO 8601 format. For example, 26th January 2014 10:32:28 in the morning with GMT+2 will be serialised as:

```
2014-01-26T10:32:28+02:00
```

API Endpoints

User Information

API	Description

GET /api/UserInfo	Get user information.
	
	Response format:
	{
	  FirstName : string,
	  MiddleName : string,
	  LastName : string,
	  DisplayedName : string,
	  AlternativeName : string,
	  EmailAddress : string,
	  UnitPreference : UnitPreferenceType,
	  SiteName : string
	}

Yacht Event Types

API	Description

GET /api/EventType	Gets a collection of maintained event types.
	
	Response format:
	{
	  EventTypeId : string,
	  Folders : string[],
	  EventValueType : EventValueType,
	  EventCategory : EventCategoryType,
	  EventName : string,
	  UnitSymbol : string,
	  MinValue : double?,
	  MaxValue : double?,
	  Choices : string[],
	  SortOrder : int
	}[]

Log Book

API	Description

GET /api/LogBook	Gets a collection of all user log books.
	
	Response format:
	{
	  LogBookKey : long,
	  Name : string
	}[]

GET /api/LogBook/{id}	Gets a specific user log book as identified by the {id}. If ID is invalid, service returns 404 Not Found. Response format: { LogBookKey : long,

	Name : string }

POST /api/LogBook	Adds a new log book.
	
	Request format:
	{
	Name : string
	}
	
	On successful creation, service returns `201 Created` with the following content.
	
	Response format:
	{
	LogBookKey : long,
	Name : string
	}

PUT /api/LogBook/{id}	Updates a specific user log book as identified by the `{id}`.
	
	If ID is invalid, service returns `404 Not Found`.
	
	Request format:
	{
	Name : string
	}

DELETE /api/LogBook/{id}	Deletes a specific user log book as identified by the `{id}`. If ID is invalid, service returns `404 Not Found`.

Log Book Segment

API	Description
GET /api/LogBookSegment	Gets a collection of all user log book segments. Response format: { LogBookSegmentKey : long, LogBookKey : long, Description : string, OriginLocationName : string, OriginTime : DateTimeOffset, DestinationLocationName : string, DestinationTime : DateTimeOffset?, Distance : double?, CrowDistance : double?, InitialRunTime : TimeSpan?, InitialTank : double?, TankConsumption : double? }[]
GET /api/LogBookSegment?logBookKey={book-key}	Gets a collection of user log book segments that are associated to a specific log book as identified by the `{book-key}`. If book key is invalid, service returns `404 Not Found`.

	Response format: { LogBookSegmentKey : long, LogBookKey : long, Description : string, OriginLocationName : string, OriginTime : DateTimeOffset, DestinationLocationName : string, DestinationTime : DateTimeOffset?, Distance : double?, CrowDistance : double?, InitialRunTime : TimeSpan?, InitialTank : double?, TankConsumption : double? }[]

GET /api/LogBookSegment/{id}	Gets a specific user log book segment as identified by the `{id}`.
	
	If ID is invalid, service returns `404 Not Found`.
	
	Response format:
	{
	LogBookSegmentKey : long,
	LogBookKey : long,
	Description : string,
	OriginLocationName : string,
	OriginTime : DateTimeOffset,
	DestinationLocationName : string,
	DestinationTime : DateTimeOffset?,
	Distance : double?,
	CrowDistance : double?,
	InitialRunTime : TimeSpan?,
	InitialTank : double?,
	TankConsumption : double?
	}

PUT /api/LogBookSegment/{id}	Updates a specific user log book segment as identified by the `{id}`.
	
	If ID is invalid, service returns `404 Not Found`.
	
	Request format:
	{
	Description : string,
	OriginLocationName : string,
	DestinationLocationName : string,
	InitialRunTime : TimeSpan?,
	InitialTank : double?,
	TankConsumption : double?
	}

DELETE /api/LogBookSegment/{id}	Deletes a specific user log book segment as identified by the `{id}`.

	If ID is invalid, service returns `404 Not Found`.

Yacht Events

API	Description

`GET /api/Event`	Gets a collection of all user yacht events.
	
	Response format:
	`{`
	`  EventKey : long,`
	`  EventTypeId : string,`
	`  Latitude : double,`
	`  Longitude : double,`
	`  Timestamp : DateTimeOffset,`
	`  Value : string`
	`}[]`

`GET /api/Event?logBookSegmentKey={segment-key}`	Gets a collection of user yacht events that are associated to a specific log book segment as identified by the `{segment-key}`.
	
	If segment key is invalid, service returns `404 Not Found`.
	
	Response format:
	`{`
	`  EventKey : long,`
	`  EventTypeId : string,`
	`  Latitude : double,`
	`  Longitude : double,`
	`  Timestamp : DateTimeOffset,`
	`  Value : string`
	`}[]`

`GET /api/Event/{id}`	Gets a specific user yacht event as identified by the `{id}`.
	
	If ID is invalid, service returns `404 Not Found`.
	
	Response format:
	`{`
	`  EventKey : long,`
	`  EventTypeId : string,`
	`  Latitude : double,`
	`  Longitude : double,`
	`  Timestamp : DateTimeOffset,`
	`  Value : string`
	`}`

`POST /api/Event`	Adds a new yacht event.
	
	Request format:
	`{`
	`  EventTypeId : string,`
	`  Latitude : double,`
	`  Longitude : double,`
	`  Timestamp : DateTimeOffset,`
	`  Value : string`
	`}`

	On successful creation, service returns `201 Created` with the following content.
	
	Response format:
	`{`
	`  EventKey : long,`
	`  EventTypeId : string,`
	`  Latitude : double,`
	`  Longitude : double,`
	`  Timestamp : DateTimeOffset,`
	`  Value : string`
	`}`

`PUT /api/Event/{id}`	Updates a specific user yacht event as identified by the `{id}`.
	
	If ID is invalid, service returns `404 Not Found`.
	
	Request format:
	`{`
	`  Latitude : double,`
	`  Longitude : double,`
	`  Timestamp : DateTimeOffset,`
	`  Value : string`
	`}`

`DELETE /api/Event/{id}`	Deletes a specific user yacht event as identified by the `{id}`. If ID is invalid, service returns `404 Not Found`.

Yacht Position

API	Description

`GET /api/Position/maxRecords={max-records}&since={since-date-time}`	Gets a collection of all user yacht positions, ordered by timestamp.
	
	The `{max-records}` is optional, and it determines the maximum number of records returned. If unspecified, the value defaults to 100. Regardless of the specified value, API will not return more than 1000 records per request.
	
	The {since-date-time} is optional. If specified, the first returned record will have a timestamp that is equal or later than the specified value. If unspecified, the first returned record has the earliest timestamp.
	
	Response format:
	`{`
	`  PositionKey : long,`
	`  Latitude : double,`
	`  Longitude : double,`
	`  Timestamp : DateTimeOffset`
	`}[]`

`GET /api/Position?logBookSegmentKey={segment-key}`	Gets a collection of user yacht positions that are associated to a specific log book segment as identified by the `{segment-key}`, ordered by timestamp.

	If segment key is invalid, service returns `404 Not Found`.
	
	Response format:
	`{`
	`  PositionKey : long,`
	`  Latitude : double,`
	`  Longitude : double,`
	`  Timestamp : DateTimeOffset`
	`}[]`

`GET /api/Position/{id}`	Gets a specific user yacht position as identified by the `{id}`.
	
	If ID is invalid, service returns `404 Not Found`.
	
	Response format:
	`{`
	`  PositionKey : long,`
	`  Latitude : double,`
	`  Longitude : double,`
	`  Timestamp : DateTimeOffset`
	`}`

`POST /api/Position`	Submits a new yacht position. When submitting a collection of positions, consider using `/api/Util/AddPositions` instead.
	
	Request format:
	`{`
	`  Latitude : double,`
	`  Longitude : double,`
	`  Timestamp : DateTimeOffset`
	`}`

`PUT /api/Position/{id}`	Updates a specific user yacht position as identified by the `{id}`.
	
	If ID is invalid, service returns `404 Not Found`.
	
	Request format:
	`{`
	`  Latitude : double,`
	`  Longitude : double,`
	`  Timestamp : DateTimeOffset`
	`}`

`DELETE /api/Position/{id}`	Deletes a specific user yacht position as identified by the `{id}`. If ID is invalid, service returns `404 Not Found`.

Yacht Properties

API	Description
`GET /api/YachtProperties`	Gets the yacht properties. Response format: `{`

	YachtName : string, Description : string, HomeLocationName : string, HomeLocationLatitude : double?, HomeLocationLongitude : double? , CallSign : string, Mmsi : string Brand : string, ModelName : string, BoatType : string, HullType : string, EngineBrand : string, Length : double? }

Log Book Settings

API	Description

GET /api/LogBookSettings	Gets the log book settings.
	
	Response format:
	{
	AutoNewBookDay: int,
	AutoNewSegmentMinute: int,
	DriftingMetre: int
	}

PUT /api/LogBookSettings	Updates the log book settings.
	
	Request format:
	{
	AutoNewBookDay: int,
	AutoNewSegmentMinute: int,
	DriftingMetre: int
	}

POST /api/LogBookSettings	Similar to the PUT request above.

Friend

API	Description

GET /api/Friend	Gets a collection of all user approved friends.
	
	Response format:
	{
	Name: string,
	EmailAddress: string,
	SiteUrl: string,
	ProfilePictureUrl: string,
	ProfilePictureThumbUrl: string,
	YachtProperties: {
	YachtName : string,
	Description : string,
	HomeLocationName : string,
	HomeLocationLatitude : double?,
	HomeLocationLongitude : double? ,
	CallSign : string,
	Mmsi : string

	Brand : string, ModelName : string, BoatType : string, HullType : string, EngineBrand : string, Length : double? } }[]

GET /api/Friend/{id}	Gets a specific user approved friend as identified by the `{id}`.
	
	If ID is invalid, service returns `404 Not Found`.
	
	Response format:
	{
	Name: string,
	EmailAddress: string,
	SiteUrl: string,
	ProfilePictureUrl: string,
	ProfilePictureThumbUrl: string,
	YachtProperties: {
	YachtName : string,
	Description : string,
	HomeLocationName : string,
	HomeLocationLatitude : double?,
	HomeLocationLongitude : double? ,
	CallSign : string,
	Mmsi : string
	Brand : string,
	ModelName : string,
	BoatType : string,
	HullType : string,
	EngineBrand : string,
	Length : double?
	}
	}

Geolocation

API	Description
GET /api/Geolocation	Gets a collection of all user geo-locations. Response format: { GeolocationKey : long Name : string Latitude : double Longitude : double Radius : double }[]
GET /api/Geolocation/{id}	Gets a specific user geo-location as identified by the `{id}`. If ID is invalid, service returns `404 Not Found`. Response format:

	{
	GeolocationKey : long
	Name : string
	Latitude : double
	Longitude : double
	Radius : double
	}

POST /api/Geolocation	Adds a geo-location.
	
	Request format:
	{
	Name : string
	Latitude : double
	Longitude : double
	Radius : double
	}
	
	On successful creation, service returns `201 Created` with the following content.
	
	Response format:
	{
	GeolocationKey : long
	Name : string
	Latitude : double
	Longitude : double
	Radius : double
	}

PUT /api/Geolocation/{id}	Updates a specific user geo-location as identified by the `{id}`.
	
	If ID is invalid, service returns `404 Not Found`.
	
	Request format:
	{
	Name : string
	Latitude : double
	Longitude : double
	Radius : double
	}

DELETE /api/Geolocation/{id}	Deletes a specific user geo-location as defined by the `{id}`. If ID is invalid, service returns `404 Not Found`.

Picture Album

API	Description
GET /api/Album	Gets a collection of all user picture albums. Response format: { PictureAlbumId : Guid AlbumName : string Order : int Accessibility : Relationship

	CreatedTime : DateTimeOffset ModifiedTime : DateTimeOffset }[]

GET /api/Album/{id}	Gets a specific user picture album as identified by the {id}.
	
	If ID is invalid, service returns 404 Not Found.
	
	Response format:
	{
	PictureAlbumId : Guid
	AlbumName : string
	Order : int
	Accessibility : Relationship
	CreatedTime : DateTimeOffset
	ModifiedTime : DateTimeOffset
	}

POST /api/Album	Adds a picture album.
	
	Request format:
	{
	AlbumName : string
	Accessibility : Relationship
	}
	
	On successful creation, service returns 201 Created with the following content.
	
	Response format:
	{
	PictureAlbumId : Guid
	AlbumName : string
	Order : int
	Accessibility : Relationship
	CreatedTime : DateTimeOffset
	ModifiedTime : DateTimeOffset
	}

PUT /api/Album/{id}	Updates a specific user picture album as identified by the {id}.
	
	If ID is invalid, service returns 404 Not Found.
	
	Request format:
	{
	AlbumName : string
	Accessibility : Relationship
	}

DELETE /api/Album/{id}	Deletes a specific user picture album as defined by the {id}. If ID is invalid, service returns 404 Not Found.

Picture

API	Description
GET /api/Picture	Gets a collection of all user pictures.

	Response format: { PictureId : Guid PictureAlbumId : Guid Caption : string Order : int Accessibility : Relationship TakenTime : DateTime? TakenLocationLatitude : double? TakenLocationLongitude : double? TakenLocationName : string LogBookSegmentKey : long? Url : string ThumbUrl : string }[]
GET /api/Picture/albumId={album-id}	Gets a collection of user pictures that are associated to a specific picture album as identified by the `{album-id}`. If album identifier is invalid, service returns `404 Not Found`. Response format: { PictureId : Guid PictureAlbumId : Guid Caption : string Order : int Accessibility : Relationship TakenTime : DateTime? TakenLocationLatitude : double? TakenLocationLongitude : double? TakenLocationName : string LogBookSegmentKey : long? Url : string ThumbUrl : string }[]
GET /api/Picture/logBookSegmentKey={segment-key}	Gets a collection of user pictures that are associated to a specific log book segment as identified by the `{segment-key}`. If segment key is invalid, service returns `404 Not Found`. Response format: { PictureId : Guid PictureAlbumId : Guid Caption : string Order : int Accessibility : Relationship TakenTime : DateTime? TakenLocationLatitude : double? TakenLocationLongitude : double?

	TakenLocationName : string LogBookSegmentKey : long? Url : string ThumbUrl : string }[]

GET /api/Picture/{id}	Gets a specific user picture as identified by the `{id}`.
	
	If ID is invalid, service returns `404 Not Found`.
	
	Response format:
	{
	PictureId : Guid
	PictureAlbumId : Guid
	Caption : string
	Order : int
	Accessibility : Relationship
	TakenTime : DateTime?
	TakenLocationLatitude : double?
	TakenLocationLongitude : double?
	TakenLocationName : string
	LogBookSegmentKey : long?
	Url : string
	ThumbUrl : string
	}

POST /api/Picture	Uploads a new picture. The content must be of MIME multipart content type.
	
	Request format:
	{
	Latitude : double,
	Longitude : double,
	Timestamp : DateTimeOffset,
	Album : Guid?,
	Picture : byte[]
	}
	
	If album identifier is not specified, the first album will be used to house the new picture. If there is no album, one album will be created.
	
	On successful creation, service returns `201 Created` with the following content.
	
	Response format:
	{
	PictureId : Guid
	PictureAlbumId : Guid
	Caption : string
	Order : int
	Accessibility : Relationship
	TakenTime : DateTime?
	TakenLocationLatitude : double?

	TakenLocationLongitude : double? TakenLocationName : string LogBookSegmentKey : long? Url : string ThumbUrl : string }
DELETE /api/Picture/{id}	Deletes a specific user picture as defined by the {id}. If ID is invalid, service returns 404 Not Found.

Utility APIs

API	Description

POST /api/Util/AddPositions	Submits a collection of new yacht positions.
	
	Request format:
	{
	Value : string
	}
	
	Value is a vertical-bar-separated string of latitude, longitude and date time values. For e.g.:
	
	1.23|3.456|2014-01-26T10:32:28+02:00
	
	-or-
	
	1.23|3.456|2014-01-26T10:32:28+02:00|1.231|3.455|2014-01-26T10:33:53+02:00
	
	Response format:
	{
	PositionsParsed : int
	}
	
	Positions parsed is the total number of positions parsed.

Appendix: Enumeration

UnitPreferenceType

Key	Value (byte)	Description
Metric	0	Unit in metric format.
US	1	Unit in US format.

EventValueType

Key	Value (byte)	Description
None	0	Event type requires no value.
Number	1	Event type accepts numerical value.
Text	2	Event type accepts textual value.
Choice	3	Event type accepts textual value, and a selection of this textual values is provided.

EventCategoryType

Key	Value (byte)	Description
Unspecified	0	
General	1	
Environmental	2	
Emergency	3	

Relationship

Key	Value (byte)	Description
Self	0	Target resource is available to resource owner.
Friend	1	Target resource is available to resource owner and approved resource owner's friends.
Public	2	Target resource is available to everyone.

ErrorCode

Key	Value (int)	Description
NoError	0	
UnknownError	1	
InvalidEmailAddress	2	
DuplicateEmailAddress	3	
LoginFailed	4	
NotLoggedIn	5	
InvalidOperation	6	
SiteNameNotAvailable	7	
InvalidAccountType	8	
EditingSuspended	9	
DuplicateYachtModel	10	
InvalidBoatType	11	
InvalidHullType	12	
InvalidYachtModel	13	
DuplicateBoatType	14	

DuplicateHullType	15	
ReferenceYachtModel	16	
ReferenceYacht	17	
InvalidYachtModelEdit	18	
InvalidPictureAlbum	19	
InvalidPicture	20	
AlbumNotEmpty	21	
InvalidLogBookSegment	22	
InvalidPictureStream	23	
InvalidConfirmationToken	24	
InvalidYachtModelLengths	25	
InvalidYachtModelWeights	26	
InvalidYachtModelSails	27	
InvalidYachtModelEngines	28	
InvalidYachtModelWaterTanks	29	
InvalidYachtModelFuelTanks	30	
InvalidYachtModelProductionYear	31	
InvalidYachtEventType	32	
InvalidLogBook	33	
LogBookNotEmpty	34	
LogBookSegmentNotClosed	35	
SegmentTimeOverlap	36	
InvalidLogBookSegmentTime	37	
InvalidEngineBrand	38	
DuplicateEngineBrand	39	
InvalidSiteTab	40	
TooManyUnapprovedUpload	41	
InvalidPassword	42	
InvalidColumnSpan	43	
DuplicateEventName	44	
InvalidUnit	45	
MissingChoices	46	
ReservedEventType	47	
ReferenceYachtEvent	48	
InvalidEventValue	49	
InvalidChoiceCharacter	50	
InvalidMinValue	51	
InvalidMaxValue	52	
DuplicateSubscriptionName	53	
DuplicateSubscriptionFeeCurrency	54	
InvalidSubscriptionFee	55	
InvalidSubscriptionChargeInterval	56	
ReferenceAccountSubscription	57	
InvalidPayPalResponse	58	
InvalidPayPalAcknowledgementCode	59	
InvalidSubscriptionPlan	60	
InvalidCoupon	61	
CouponNotApplicable	62	
MissingNextBillDate	63	
ProFeatureAccessOnly	64	

InvalidUser	65	
InvalidExternalLogger	66	
ValueRequired	67	
DuplicateCouponCode	68	
InvalidValue	69	
EndDateIsEarlierThanStartDate	70	
DuplicateTabTitle	71	
ConsecutiveSegmentsRequired	72	
DuplicateYachtBrand	73	
InvalidYachtBrand	74	
InvalidYachtEvent	75	
InvalidDeviceId	76	
InvalidEventName	77	