Members

***AnyBoard* : `object`**

Global variable AnyBoard.

Typedefs

***playDrawCallback* : `function`**

This type of callback will be called when card is drawn or played

***simpleTriggerCallback* : `function`**

Type of callback called upon triggering of events

***stdStringCallback* : `function`**

Generic callback returning a string param

***stdBoolCallback* : `function`**

Generic callback returning a bool param

***stdNoParamCallback* : `function`**

Generic callback without params

***onScanCallback* : `function`**

Type of callback called upon detecting a token

***stdErrorCallback* : `function`**

This type of callback will be called upon failure to complete a function

AnyBoard : `object`

Global variable AnyBoard.

Kind: global variable

- AnyBoard : `object`
  - .Driver
    - new AnyBoard.Driver(options)
    - .toString() ⇒ `string`
  - .Deck
    - new AnyBoard.Deck(name, jsonDeck)
    - *instance*
      - .shuffle()
      - .initiate(jsonDeck)
      - .refill([newDeck])
      - .onPlay(func)
      - .onDraw(callback)
      - .toString() ⇒ `string`
    - *static*
      - .get(name) ⇒ `Deck`
  - .Card
    - new AnyBoard.Card(deck, options)
    - *instance*
      - .onPlay(func)
      - .onDraw(callback)
      - .toString() ⇒ `string`

    - *static*
      - .get(cardTitleOrID) ⇒ `Card`
  - .Dices
    - new AnyBoard.Dices([eyes], [numOfDice])
    - .roll() ⇒ `number`
    - .rollEach() ⇒ `Array`
  - .Player
    - new AnyBoard.Player(name, [options])
    - *instance*
      - .pay(resources, [receivingPlayer]) ⇒ `boolean`
      - .trade(giveResources, receiveResources, [player]) ⇒ `boolean`
      - .recieve(resourceSet)
      - .draw(deck, [options]) ⇒ `Card`
      - .play(card, [customOptions]) ⇒ `boolean`
      - .toString() ⇒ `string`
    - *static*
      - .get(name) ⇒ `Player`
  - .Hand
    - new AnyBoard.Hand(player, [options])
    - .has(card, [amount]) ⇒ `boolean`
    - .discardHand()
    - .discardCard(card)
    - .toString() ⇒ `string`
  - .Resource
    - new AnyBoard.Resource(name, [properties])
    - .get(name) ⇒ `Resource`
  - .ResourceSet
    - new AnyBoard.ResourceSet([resources], [allowNegative])
    - .contains(reqResource) ⇒ `boolean`
    - .add(resourceSet)
    - .subtract(resourceSet) ⇒ `boolean`
    - .similarities(resourceSet) ⇒ `object`
  - .BaseToken
    - new AnyBoard.BaseToken(name, address, device, [driver])
    - *instance*
      - .isConnected() ⇒ `boolean`
      - .connect([win], [fail])
      - .disconnect()
      - .trigger(eventName, [eventOptions])
      - .on(eventName, callbackFunction)
      - .once(eventName, callbackFunction)
      - .send(data, [win], [fail])
      - .print(value, [win], [fail])
      - .getFirmwareName([win], [fail])
      - .getFirmwareVersion([win], [fail])
      - .getFirmwareUUID([win], [fail])
      - .hasLed([win], [fail])
      - .hasLedColor([win], [fail])
      - .hasVibration([win], [fail])
      - .hasColorDetection([win], [fail])
      - .hasLedScreen([win], [fail])
      - .hasRfid([win], [fail])
      - .hasNfc([win], [fail])
      - .hasAccelometer([win], [fail])
      - .hasTemperature([win], [fail])
      - .ledOn(value, [win], [fail])

        - .ledBlink(value, [win], [fail])
        - .ledOff([win], [fail])
        - .toString() ⇒ `string`
      - *static*
        - .setDefaultDriver(driver) ⇒ `boolean`
    - .Drivers : `Object`
      - .get(name) ⇒ `Driver` | `undefined`
      - .getCompatibleDriver(type, compatibility) ⇒ `Driver`
    - .TokenManager
      - .setDriver(driver)
      - .scan([win], [fail], [timeout])
      - .get(address) ⇒ `BaseToken`
    - .Logger
      - .warn(message, [sender])
      - .error(message, [sender])
      - .log(message, [sender])
      - .debug(message, [sender])
      - .setThreshold(severity)
    - .Utils
      - .isEqual(a, b, [aStack], [bStack]) ⇒ `boolean`

AnyBoard.Driver

Kind: static class of `AnyBoard`
Properties

Name	Type	Description
name	`string`	name of the driver
description	`string`	description of the driver
version	`string`	version of the driver
dependencies	`string`	Text describing what, if anything, the driver depends on.
date	`string`	Date upon release/last build.
type	`Array`	Array of string describing Type of driver, e.g. "bluetooth"
compatibility	`Array` | `object` | `string`	An object or string that can be used to deduce compatibiity, or an array of different compatibilies.
properties	`object`	dictionary that holds custom attributes

- .Driver
  - new AnyBoard.Driver(options)
  - .toString() ⇒ `string`

new AnyBoard.Driver(options)

Represents a single Driver, e.g. for spesific token or bluetooth discovery

Param	Type	Description
options	`object`	options for the driver
options.name	`string`	name of the driver
options.description	`string`	description of the driver
options.version	`string`	version of the driver

options.type	`string`	Type of driver, e.g. "bluetooth"
options.compatibility	`Array` | `object` | `string`	An object or string that can be used to deduce compatibiity, or an array of different compatibilies. How this is used is determined by the set standard driver on TokenManager that handles scanning for and connecting to tokens.
[options.dependencies]	`string`	*(optional)* What if anything the driver depends on.
[options.date]	`string`	*(optional)* Date upon release/last build.
options.yourAttributeHere	`any`	custom attributes, as well as specified ones, are all placed in driver.properties. E.g. 'heat' would be placed in driver.properties.heat.

driver.toString() ⇒ `string`

Returns a short description of the Driver instance

Kind: instance method of `Driver`

AnyBoard.Deck

Kind: static class of `AnyBoard`
Properties

Name	Type	Description
name	`string`	name of Deck.
cards	`Array.<Card>`	complete set of cards in the deck
pile	`Array.<Card>`	remaining cards in this pile
usedPile	`Array.<Card>`	cards played from this deck
autoUsedRefill	`boolean`	*(default: true)* whether or not to automatically refill pile from usedPile when empty. Is ignored if autoNewRefill is true.
autoNewRefill	`boolean`	*(default: false)* whether or not to automatically refill pile with a whole new deck when empty.
playListeners	`Array.<function()>`	holds functions to be called when cards in this deck are played
drawListeners	`Array.<function()>`	holds functions to be called when cards in this deck are drawn

new AnyBoard.Deck(name, jsonDeck)

Represents a Deck of Cards

Param	Type	Description
name	`string`	name of Deck. This name can be used to retrieve the deck via AnyBoard.Deck.all[name].
jsonDeck	`object`	loaded JSON file. See examples/deck-loading/ for JSON format and loading.

deck.shuffle()

Shuffles the pile of undrawn cards . Pile is automatically shuffled upon construction, and upon initiate(). New cards added upon refill() are also automatically shuffled.

Kind: instance method of `Deck`

deck.initiate(jsonDeck)

Reads Deck from jsonObject and provides a shuffled version in pile. Is automatically called upon constructing a deck.

Kind: instance method of `Deck`

Param	Type	Description
jsonDeck	`object`	loaded json file. See examples-folder for example of json file and loading

deck.refill([newDeck])

Manually refills the pile. This is not necessary if autoUsedRefill or autoNewRefill property of deck is true.

Kind: instance method of `Deck`

Param	Type	Default	Description
[newDeck]	`boolean`	`false`	*(default: false)* True if to refill with a new deck. False if to refill with played cards (from usedPile)

deck.onPlay(func)

Adds functions to be executed upon all Cards in this Deck.

Kind: instance method of `Deck`

Param	Type	Description
func	`playDrawCallback`	callback function to be executed upon play of card from this deck

deck.onDraw(callback)

Adds functions to be executed upon draw of Card from this Deck

Kind: instance method of `Deck`

Param	Type	Description
callback	`playDrawCallback`	function to be executed with the 3 parameters AnyBoard.Card, AnyBoard.Player, (options) when cards are drawn

deck.toString() ⇒ `string`

Sting representation of a deck

Kind: instance method of `Deck`

Deck.get(name) ⇒ `Deck`

Returns deck with given name

Kind: static method of `Deck`
Returns: `Deck` - deck with given name (or undefined if non-existent)

Param	Type	Description
name	`string`	name of deck

AnyBoard.Card

Kind: static class of `AnyBoard`
Properties

Name	Type	Description
title	`string`	title of the card.
description	`string`	description for the Card
color	`string`	color of the Card
category	`string`	category of the card, not used by AnyBoard FrameWork
value	`number`	value of the card, not used by AnyBoard FrameWork
type	`string`	type of the card, not used by AnyBoard FrameWork
amount	`number`	amount of this card its deck
deck	`Deck`	deck that this card belongs to
playListeneres	`Array`	holds functions to be called upon play of this spesific card (before potential playListeners on its belonging deck)
drawListeners	`Array`	holds functions to be called upon draw of this spesific card (before potential drawListeners on its belonging deck)
properties	`object`	dictionary that holds custom attributes

new AnyBoard.Card(deck, options)

Represents a single Card Should be instantiated in bulk by calling the deck constructor

Param	Type	Default	Description
deck	`Deck`		deck to which the card belongs
options	`object`		options for the card
options.title	`string`		title of the card.

options.description	`string`		description for the Card
[options.color]	`string`		*(optional)* color of the Card
[options.category]	`string`		*(optional)* category of the card, not used by AnyBoard FrameWork
[options.value]	`number`		*(optional)* value of the card, not used by AnyBoard FrameWork
[options.type]	`string`		*(optional)* type of the card, not used by AnyBoard FrameWork
[options.amount]	`number`	`1`	*(optional, default: 1)* amount of this card in the deck
[options.yourAttributeHere]	`any`		custom attributes, as well as specified ones, are all placed in card.properties. E.g. 'heat' would be placed in card.properties.heat.

card.onPlay(func)

Adds functions to be executed upon a play of this card

Kind: instance method of `Card`

Param	Type	Description
func	`playDrawCallback`	callback function to be executed upon play of card from this deck

card.onDraw(callback)

Adds functions to be executed upon a draw of this card

Kind: instance method of `Card`

Param	Type	Description
callback	`playDrawCallback`	function to be executed upon play of card from this deck

card.toString() ⇒ `string`

Returns a string representation of the card.

Kind: instance method of `Card`

Card.get(cardTitleOrID) ⇒ `Card`

Returns card with given id

Kind: static method of `Card`
Returns: `Card` - card with given id (or undefined if non-existent)

Param	Type	Description
cardTitleOrID	`number` | `string`	id or title of card

AnyBoard.Dices

Kind: static class of `AnyBoard`

- .Dices
  - new AnyBoard.Dices([eyes], [numOfDice])
  - .roll() ⇒ `number`
  - .rollEach() ⇒ `Array`

new AnyBoard.Dices([eyes], [numOfDice])

Represents a set of game dices that can be rolled to retrieve a random result.

Param	Type	Default	Description
[eyes]	`number`	`6`	*(default: 6)* number of max eyes on a roll with this dice
[numOfDice]	`number`	`1`	*(default: 1)* number of dices

Example

```
// will create 1 dice, with 6 eyes
var dice = new AnyBoard.Dices();

// will create 2 dice, with 6 eyes
var dice = new AnyBoard.Dices(2, 6);
```

dices.roll() ⇒ `number`

Roll the dices and returns a the sum

Kind: instance method of `Dices`
Returns: `number` - combined result of rolls for all dices
Example

```
var dice = new AnyBoard.Dices();

// returns random number between 1 and 6
dice.roll()
```

Example

```
var dice = new AnyBoard.Dices(2, 6);

// returns random number between 1 and 12
dice.roll()
```

dices.rollEach() ⇒ `Array`

Roll the dices and returns an array of results for each dice

Kind: instance method of `Dices`
Returns: `Array` - list of results for each dice
Example

```
var dice = new AnyBoard.Dices(2, 6);

// returns an Array of numbers
var resultArray = dice.rollEach()

// result of first dice, between 1-6
resultArray[0]

// result of second dice, between 1-6
resultArray[1]
```

AnyBoard.Player

Kind: static class of `AnyBoard`

Properties

Name	Type	Description
hand	`Hand`	hand of cards (Quests)
faction	`string`	faction (Sp[ecial abilities or perks)
class	`string`	class (Special abilities or perks)
holds	`ResourceSet`	the resources belonging to this player
color	`string`	color representation of player

new AnyBoard.Player(name, [options])

Represents a Player (AnyBoard.Player)

Param	Type	Description
name	`string`	name of the player
[options]	`object`	*(optional)* options for the player
[options.color]	`string`	*(optional)* color representing the player
[options.faction]	`string`	*(optional)* faction representing the player
[options.class]	`string`	*(optional)* class representing the player
[options.yourAttributeHere]	`any`	*(optional)* custom attributes, as well as specified ones, are all placed in player.properties. E.g. 'age' would be placed in player.properties.age.

player.pay(resources, [receivingPlayer]) ⇒ `boolean`

Take resources from this player and give to receivingPlayer.

Kind: instance method of `Player`
Returns: `boolean` - whether or not transaction was completed (false if Player don't hold enough resources)

Param	Type	Description
resources	`ResourceSet`	dictionary of resources
[receivingPlayer]	`Player`	*(optional)* Who shall receive the resources. Omit if not to anyone (e.g. give to "the bank")

player.trade(giveResources, receiveResources, [player]) ⇒ `boolean`

Trade resources between players/game

Kind: instance method of `Player`

Returns: `boolean` - whether or not transaction was completed (false if Player don't hold enough resources)

Param	Type	Description
giveResources	`ResourceSet`	resources this player shall give
receiveResources	`ResourceSet`	resources this player receieves
[player]	`Player`	*(optional)* Who shall be traded with. Omit if not to a player, but to "the bank".

Example

```
new AnyBoard.Resource("gold");
new AnyBoard.Resource("silver");

var startTreasure = new AnyBoard.ResourceSet({"gold": 6, "silver": 42});
var goldTreasure = new AnyBoard.ResourceSet({"gold": 2});
var silverTreasure = new AnyBoard.ResourceSet({"silver": 12});

var dr1 = new AnyBoard.Player("firstDoctor");
var dr2 = new AnyBoard.Player("secondDoctor");

dr1.receive(startTreasure);
dr2.receive(startTreasure);

// returns true. dr1 will now own {"gold": 4, "silver": 54}. dr2 owns {"gold": 8, "silver": 30}
dr1.trade(goldTreasure, silverTreasure, dr2)
```

Example

```
// returns true. dr1 will now own {"gold": 2, "silver": 66}. dr2 still owns {"gold": 8, "silver": 30}
dr1.trade(goldTreasure, silverTreasure)
```

Example

```
var firstOverlappingTreasure = new AnyBoard.ResourceSet({"silver": 115, "gold": "6"});
var secondOverlappingTreasure= new AnyBoard.ResourceSet({"silver": 100, "gold": "7"});

// returns true. The trade nullifies the similarities, so that the trade can go through even though
//     dr1 has < 100 silver
dr1.trade(firstOverlappingTreasure, secondOverlappingTreasure)
```

player.recieve(resourceSet)

Receive resource from bank/game. Use pay() when receiving from players.

Kind: instance method of `Player`

Param	Type	Description
resourceSet	`ResourceSet`	resources to be added to this players bank

Example

```
new AnyBoard.Resource("gold");
new AnyBoard.Resource("silver");

var startTreasure = new AnyBoard.ResourceSet({"gold": 6, "silver": 42});
var secondTresure = new AnyBoard.ResourceSet({"silver": 12, "copper": 122});

var dr1 = new AnyBoard.Player("firstDoctor");  // player owns nothing initially
```

```
dr1.receive(startTreasure);  // owns {"gold": 6, "silver": 42}
dr1.receive(secondTresure);  // owns {"gold": 6, "silver": 54, "copper": 122}
```

player.draw(deck, [options]) ⇒ `Card`

Draws a card from a deck and puts it in the hand of the player

Kind: instance method of `Player`
Returns: `Card` - card that is drawn

Param	Type	Description
deck	`Deck`	deck to be drawn from
[options]	`object`	*(optional)* parameters to be sent to the drawListeners on the deck

Example

```
var dr1 = new AnyBoard.Player("firstDoctor");  // player has no cards initially

// Now has one card
dr1.draw(deck);

// Now has two cards. option parameter is being passed on to any drawListeners (See Deck/Card)
dr1.draw(deck, options);
```

player.play(card, [customOptions]) ⇒ `boolean`

Plays a card from the hand. If the hand does not contain the card, the card is not played and the hand unchanged.

Kind: instance method of `Player`
Returns: `boolean` - whether or not the card was played

Param	Type	Description
card	`Card`	card to be played
[customOptions]	`object`	*(optional)* custom options that the play should be played with

Example

```
var DrWho = new AnyBoard.Player("firstDoctor");  // player has no cards initially

// Store the card that was drawn
var card = DrWho.draw(existingDeck);

// Play that same card
DrWho.play(card)
```

player.toString() ⇒ `string`

Returns a string representation of the player

Kind: instance method of `Player`

Player.get(name) ⇒ `Player`

Returns player with given name

Kind: static method of `Player`
Returns: `Player` - player with given name (or undefined if non-existent)

Param	Type	Description
name	`string`	name of player

AnyBoard.Hand

Kind: static class of `AnyBoard`

new AnyBoard.Hand(player, [options])

Represents a Hand of a player, containing cards. Players are given one Hand in Person constructor.

Param	Type	Description
player	`Player`	player to which this hand belongs
[options]	`object`	*(optional)* custom properties added to this hand

hand.has(card, [amount]) ⇒ `boolean`

Checks whether or not a player has an amount card in this hand.

Kind: instance method of `Hand`

Returns: `boolean` - hasCard whether or not the player has that amount or more of that card in this hand

Param	Type	Default	Description
card	`Card`		card to be checked if is in hand
[amount]	`number`	`1`	*(default: 1)* amount of card to be checked if is in hand

Example

```
var DrWho = new AnyBoard.Player("firstDoctor");  // player has no cards initially

// Store the card that was drawn
var tardis = DrWho.draw(tardisDeck);

// returns true
DrWho.hand.has(card)

// returns false, as he has only one
DrWho.hand.has(card, 3)
```

hand.discardHand()

Discard the entire hand of the player, leaving him with no cards

Kind: instance method of `Hand`

hand.discardCard(card)

Discard a card from the hand of the player

Kind: instance method of `Hand`

Param	Type	Description
card	`Card`	card to be discarded.

hand.toString() ⇒ `string`

Returns a string representation of the hand

Kind: instance method of `Hand`

AnyBoard.Resource

Kind: static class of `AnyBoard`
Properties

Name	Type	Description
name	`string`	name of resource
properties	`any`	custom options added to resource

new AnyBoard.Resource(name, [properties])

Represents a simple resource (AnyBoard.Resource)

Param	Type	Description
name	`string`	name representing the resource
[properties]	`object`	*(optional)* custom properties of this resource

Example

```
var simpleGold = new AnyBoard.Resource("gold");

// The optional properties parameter can be of any type.
var advancedPowder = new AnyBoard.Resource("powder", {"value": 6, "color": "blue"});

// 6
advancedPowder.properties.value
```

Resource.get(name) ⇒ `Resource`

Returns resource with given name

Kind: static method of `Resource`
Returns: `Resource` - resource with given name (or undefined if non-existent)

Param	Type	Description
name	`string`	name of resource

Example

```
var simpleGold = new AnyBoard.Resource("gold");

// returns simpleGold
AnyBoard.Resource.get("gold");
```

AnyBoard.ResourceSet

Kind: static class of `AnyBoard`

Properties

Name	Type	Default	Description
resources	`object`		*(optional)* a set of initially contained resources
allowNegative	`boolean`	`false`	*(default: false)* whether or not to allow being subtracted resources to below 0 (dept)

new AnyBoard.ResourceSet([resources], [allowNegative])

Creates a ResourceSet

Param	Type	Default	Description
[resources]	`object`		*(optional)* a set of initially contained resources
[allowNegative]	`boolean`	`false`	*(default: false)* whether or not to allow being subtracted resources to below 0 (dept)

Example

```
// Returns a resourceset that can be deducted below 0
var debtBank = new AnyBoard.ResourceSet({}, true);
```

resourceSet.contains(reqResource) ⇒ `boolean`

Whether or not a ResourceSet contains another ResourceSet

Kind: instance method of `ResourceSet`

Returns: `boolean` - true if this ResourceSet contains reqResource, else false

Param	Type	Description
reqResource	`ResourceSet`	ResourceSet to be compared against

Example

```
new AnyBoard.Resource("gold");
new AnyBoard.Resource("silver");

var myTreasure = new AnyBoard.ResourceSet({"gold": 6, "silver": 42});
var minorDebt = new AnyBoard.ResourceSet({"gold": 1, "silver": 3});
var hugeDebt = new AnyBoard.ResourceSet({"gold": 12, "silver": 41});

// returns true
myTreasure.contains(minorDebt);

// returns false
myTreasure.contains(hugeDebt);
```

resourceSet.add(resourceSet)

Adds a ResourceSet to this one

Kind: instance method of `ResourceSet`

Param	Type	Description
resourceSet	`ResourceSet`	ResourceSet to be added to this one

Example

```
new AnyBoard.Resource("gold");
new AnyBoard.Resource("silver");

var myTreasure = new AnyBoard.ResourceSet({"gold": 6, "silver": 42});
var minorGift = new AnyBoard.ResourceSet({"silver": 2});

myTreasure.add(minorGift);
// myTreasure is now {"gold": 6, "silver": 45}
```

resourceSet.subtract(resourceSet) ⇒ `boolean`

Subtracts a dictionary of resources and amounts to a ResourceSet

Kind: instance method of `ResourceSet`
Returns: `boolean` - whether or not resources were subtracted successfully

Param	Type	Description
resourceSet	`ResourceSet`	set of resources to be subtracted

Example

```
new AnyBoard.Resource("gold");
new AnyBoard.Resource("silver");

var myTreasure = new AnyBoard.ResourceSet({"gold": 6, "silver": 42});
var minorGift = new AnyBoard.ResourceSet({"silver": 2});
var debtBank = new AnyBoard.ResourceSet({}, true);
var cosyBank = new AnyBoard.ResourceSet();

// returns true. myTreasure becomes {"gold": 6, "silver": 40}
myTreasure.subtract(minorGift);

// returns true. debtbank becomes {"silver": -2}
debtBank.subtract(minorGift);

// returns false and leaves cosyBank unchanged
cosyBank.subtract(minorGift);
```

resourceSet.similarities(resourceSet) ⇒ `object`

Returns the common resources and minimum amount between a dictionary of resources and amounts, and this ResourceSet

Kind: instance method of `ResourceSet`
Returns: `object` - similarities dictionary of common resources and amounts

Param	Type	Description
resourceSet	`ResourceSet`	dictionary of resources and amounts to be compared against

Example

```
new AnyBoard.Resource("gold");
new AnyBoard.Resource("silver");

var myTreasure = new AnyBoard.ResourceSet({"gold": 6, "silver": 42});
var otherTresure = new AnyBoard.ResourceSet({"silver": 2, "bacon": 12});

// returns {"silver": 2}
myTreasure.similarities(otherTresure);
```

AnyBoard.BaseToken

Kind: static class of `AnyBoard`

Properties

Name	Type	Description
name	`string`	name of the token
address	`string`	address of the token found when scanned
connected	`boolean`	whether or not the token is connected
device	`object`	driver spesific data.
listeners	`object`	functions to be execute upon certain triggered events
onceListeners	`object`	functions to be execute upon next triggering of certain events
sendQueue	`Array.<function()>`	queue for communicating with
cache	`object`	key-value store for caching certain communication calls
driver	`Driver`	driver that handles communication

- .BaseToken
  - new AnyBoard.BaseToken(name, address, device, [driver])
  - *instance*
    - .isConnected() ⇒ `boolean`
    - .connect([win], [fail])
    - .disconnect()
    - .trigger(eventName, [eventOptions])
    - .on(eventName, callbackFunction)
    - .once(eventName, callbackFunction)
    - .send(data, [win], [fail])
    - .print(value, [win], [fail])
    - .getFirmwareName([win], [fail])
    - .getFirmwareVersion([win], [fail])
    - .getFirmwareUUID([win], [fail])
    - .hasLed([win], [fail])
    - .hasLedColor([win], [fail])
    - .hasVibration([win], [fail])
    - .hasColorDetection([win], [fail])
    - .hasLedScreen([win], [fail])
    - .hasRfid([win], [fail])
    - .hasNfc([win], [fail])
    - .hasAccelometer([win], [fail])
    - .hasTemperature([win], [fail])
    - .ledOn(value, [win], [fail])
    - .ledBlink(value, [win], [fail])

- .ledOff([win], [fail])
- .toString() ⇒ `string`

- *static*
  - .setDefaultDriver(driver) ⇒ `boolean`

new AnyBoard.BaseToken(name, address, device, [driver])

Base class for tokens. Should be used by communication driver upon AnyBoard.TokenManager.scan()

Param	Type	Default	Description
name	`string`		name of the token
address	`string`		address of the token found when scanned
device	`object`		device object used and handled by driver
[driver]	`Driver`	`AnyBoard.BaseToken._defaultDriver`	token driver for handling communication with it.

baseToken.isConnected() ⇒ `boolean`

Returns whether or not the token is connected

Kind: instance method of `BaseToken`
Returns: `boolean` - true if connected, else false

baseToken.connect([win], [fail])

Attempts to connect to token. Uses TokenManager driver, not its own, since connect needs to happen before determining suitable driver.

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdNoParamCallback`	*(optional)* function to be executed upon success
[fail]	`stdErrorCallback`	*(optional)* function to be executed upon failure

baseToken.disconnect()

Disconnects from the token.

Kind: instance method of `BaseToken`

baseToken.trigger(eventName, [eventOptions])

Trigger an event on a token

Kind: instance method of `BaseToken`

Param	Type	Description
eventName	`string`	name of event
[eventOptions]	`object`	*(optional)* dictionary of parameters and values

Example

```
var onTimeTravelCallback = function (options) {console.log("The tardis is great!")};
existingToken.on('timeTravelled', onTimeTravelCallback);

// Triggers the function, and prints praise for the tardis
existingToken.trigger('timeTravelled');
```

```
existingToken.trigger('timeTravelled');  // prints again
existingToken.trigger('timeTravelled');  // prints again
```

baseToken.on(eventName, callbackFunction)

Adds a callbackFunction to be executed always when event is triggered

Kind: instance method of `BaseToken`

Param	Type	Description
eventName	`string`	name of event to listen to
callbackFunction	`simpleTriggerCallback`	function to be executed

Example

```
var onTimeTravelCallback = function () {console.log("The tardis is great!")};
existingToken.on('timeTravelled', onTimeTravelCallback);

// Triggers the function, and prints praise for the tardis
existingToken.trigger('timeTravelled');

existingToken.trigger('timeTravelled');  // prints again
existingToken.trigger('timeTravelled');  // prints again
```

Example

```
var onTimeTravelCallback = function (options) {
    // Options can be left out of a trigger. You should therefore check
    // that input is as expected, throw an error or give a default value
    var name = (options && options.name ? options.name : "You're");

    console.log(options.name + " is great!");
};
existingToken.on('timeTravelled', onTimeTravelCallback);

// prints "Dr.Who is great!"
existingToken.trigger('timeTravelled', {"name": "Dr.Who"});

// prints "You're great!"
existingToken.trigger('timeTravelled');
```

baseToken.once(eventName, callbackFunction)

Adds a callbackFunction to be executed next time an event is triggered

Kind: instance method of `BaseToken`

Param	Type	Description
eventName	`string`	name of event to listen to
callbackFunction	`simpleTriggerCallback`	function to be executed

Example

```
var onTimeTravelCallback = function (options) {console.log("The tardis is great!")};
existingToken.once('timeTravelled', onTimeTravelCallback);

// Triggers the function, and prints praise for the tardis
existingToken.trigger('timeTravelled');
```

```
// No effect
existingToken.trigger('timeTravelled');
```

baseToken.send(data, [win], [fail])

Sends data to the token. Uses either own driver, or (if not set) TokenManager driver

Kind: instance method of `BaseToken`

Param	Type	Description
data	`Uint8Array` | `ArrayBuffer` | `String`	data to be sent
[win]	`stdNoParamCallback`	*(optional)* function to be executed upon success
[fail]	`stdErrorCallback`	*(optional)* function to be executed upon error

baseToken.print(value, [win], [fail])

Prints to Token

String can have special tokens to signify some printer command, e.g. ##n = newLine. Refer to the individual driver for token spesific implementation and capabilites

Kind: instance method of `BaseToken`

Param	Type	Description
value	`string`	
[win]	`stdNoParamCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.getFirmwareName([win], [fail])

Gets the name of the firmware type of the token

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdStringCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

Example

```
// Function to be executed upon name retrieval
var getNameCallback = function (name) {console.log("Firmware name: " + name)};

// Function to be executed upon failure to retrieve name
var failGettingNameCallback = function (name) {console.log("Couldn't get name :(")};

existingToken.getFirmwareName(getNameCallback, failGettingNameCallback);

// Since it's asyncronous, this will be printed before the result
console.log("This comes first!")
```

baseToken.getFirmwareVersion([win], [fail])

Gets the version of the firmware type of the token

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdStringCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.getFirmwareUUID([win], [fail])

Gets a uniquie ID the firmware of the token

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdStringCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.hasLed([win], [fail])

Checks whether or not the token has simple LED

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdBoolCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.hasLedColor([win], [fail])

Checks whether or not the token has colored LEDs

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdBoolCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.hasVibration([win], [fail])

Checks whether or not the token has vibration

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdBoolCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.hasColorDetection([win], [fail])

Checks whether or not the token has ColorDetection

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdBoolCallback`	*(optional)* callback function to be called upon successful execution

[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.hasLedScreen([win], [fail])

Checks whether or not the token has LedSceen

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdBoolCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.hasRfid([win], [fail])

Checks whether or not the token has RFID reader

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdBoolCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.hasNfc([win], [fail])

Checks whether or not the token has NFC reader

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdBoolCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.hasAccelometer([win], [fail])

Checks whether or not the token has Accelometer

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdBoolCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.hasTemperature([win], [fail])

Checks whether or not the token has temperature measurement

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdBoolCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon failure

baseToken.ledOn(value, [win], [fail])

Sets color on token

Kind: instance method of `BaseToken`

Param	Type	Description
value	`string` | `Array`	string with color name or array of [red, green, blue] values 0-255
[win]	`stdNoParamCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon

Example

```
// sets Led to white
existingToken.ledOn([255, 255, 255]);

// sets Led to white (See driver implementation for what colors are supported)
existingToken.ledOn("white");
```

baseToken.ledBlink(value, [win], [fail])

tells token to blink its led

Kind: instance method of `BaseToken`

Param	Type	Description
value	`string` | `Array`	string with color name or array of [red, green, blue] values 0-255
[win]	`stdNoParamCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon

Example

```
// blinks red
existingToken.ledBlink([255, 0, 0]);

// blinks blue
existingToken.ledBlink("blue");
```

baseToken.ledOff([win], [fail])

Turns LED off

Kind: instance method of `BaseToken`

Param	Type	Description
[win]	`stdNoParamCallback`	*(optional)* callback function to be called upon successful execution
[fail]	`stdErrorCallback`	*(optional)* callback function to be executed upon

baseToken.toString() ⇒ `string`

Representational string of class instance.

Kind: instance method of `BaseToken`

BaseToken.setDefaultDriver(driver) ⇒ `boolean`

Sets a new default driver to handle communication for tokens without specified driver. The driver must have implement

a method *send(win, fail)* in order to discover tokens.

Kind: static method of `BaseToken`
Returns: `boolean` - whether or not driver was successfully set

Param	Type	Description
driver	`Driver`	driver to be used for communication

AnyBoard.Drivers : `Object`

Manager of drivers.

Kind: static property of `AnyBoard`

- .Drivers : `Object`
  - .get(name) ⇒ `Driver` | `undefined`
  - .getCompatibleDriver(type, compatibility) ⇒ `Driver`

Drivers.get(name) ⇒ `Driver` | `undefined`

Returns driver with given name

Kind: static method of `Drivers`
Returns: `Driver` | `undefined` - driver with given name (or undefined if non-existent)

Param	Type	Description
name	`string`	name of driver

Example

```
var discoveryBluetooth = new AnyBoard.Driver({
        name: 'theTardisMachine',
        description: 'bla bla',
        version: '1.0',
        type: ['bluetooth-discovery', 'bluetooth'],
        compatibility: ['tardis', 'pancakes']
    });

// Returns undefined
AnyBoard.Drivers.get("non-existant-driver")

// Returns driver
AnyBoard.Drivers.get("theTardisMachine")
```

Drivers.getCompatibleDriver(type, compatibility) ⇒ `Driver`

Returns first driver of certain type that matches the given compatibility.

Kind: static method of `Drivers`
Returns: `Driver` - compatible driver (or undefined if non-existent)

Param	Type	Description
type	`string`	name of driver
compatibility	`string`	`object`	name of driver

Example

```
var discoveryBluetooth = new AnyBoard.Driver({
```

```
        name: 'theTardisMachine',
        description: 'bla bla',
        version: '1.0',
        type: ['bluetooth-discovery', 'bluetooth'],
        compatibility: ['tardis', {"show": "Doctor Who"}]
    });

// Returns undefined (right type, wrong compatibility)
AnyBoard.Drivers.getCompatibleDriver('bluetooth', 'weirdCompatibility');

// Returns undefined (wrong type, right compatibility)
AnyBoard.Drivers.getCompatibleDriver('HTTP, {"service": "iCanTypeAnyThingHere"});

// Returns discoveryBluetooth driver
AnyBoard.Drivers.getCompatibleDriver('bluetooth', 'tardis');
```

AnyBoard.TokenManager

A token manager. Holds all tokens. Discovers and connects to them.

Kind: static property of `AnyBoard`

Properties

Name	Type	Description
tokens	`object`	dictionary of connect tokens that maps id to object
driver	`Driver`	driver for communication with tokens. Set with setDriver(driver);

TokenManager.setDriver(driver)

Sets a new default driver to handle communication for tokens without specified driver. The driver must have implemented methods *scan(win, fail, timeout) connect(token, win, fail) and disconnect(token, win, fail)*, in order to discover tokens.

Kind: static method of `TokenManager`

Param	Type	Description
driver	`Driver`	driver to be used for communication

TokenManager.scan([win], [fail], [timeout])

Scans for tokens nearby and stores in discoveredTokens property

Kind: static method of `TokenManager`

Param	Type	Description
[win]	`onScanCallback`	*(optional)* function to be executed when devices are found (called for each device found)
[fail]	`stdErrorCallback`	*(optional)* function to be executed upon failure
[timeout]	`number`	*(optional)* amount of milliseconds to scan before stopping. Driver has a default.

Example

```
var onDiscover = function(token) { console.log("I found " + token) };

// Scans for tokens. For every token found, it prints "I found ...")
TokenManager.scan(onDiscover);
```

TokenManager.get(address) ⇒ `BaseToken`

Returns a token handled by this TokenManager

Kind: static method of `TokenManager`
Returns: `BaseToken` - token if handled by this tokenManager, else undefined

Param	Type	Description
address	`string`	identifer of the token found when scanned

AnyBoard.Logger

Static logger object that handles logging. Will log using hyper.log if hyper is present (when using Evothings). Will then log all events, regardless of severity

Kind: static property of `AnyBoard`
Properties

Name	Type	Description
threshold	`number`	*(default: 10)* threshold on whether or not to log an event. Any message with level above or equal threshold will be logged
debugLevel	`number`	*(value: 0)* sets a threshold for when a log should be considered a debug log event.
normalLevel	`number`	*(value: 10)* sets a threshold for when a log should be considered a normal log event.
warningLevel	`number`	*(value: 20)* sets a threshold for when a log should be considered a warning.
errorLevel	`number`	*(value: 30)* sets a threshold for when a log should be considered a fatal error.
loggerObject	`object`	*(default: console)* logging method. Must have implemented .debug(), .log(), .warn() and .error()

Logger.warn(message, [sender])

logs a warning. Ignored if threshold > this.warningLevel (default: 20)

Kind: static method of `Logger`

Param	Type	Description
message	`string`	event to be logged
[sender]	`object`	*(optional)* sender of the message

Logger.error(message, [sender])

logs an error. Will never be ignored.

Kind: static method of `Logger`

Param	Type	Description
message	`string`	event to be logged
[sender]	`object`	*(optional)* sender of the message

Logger.log(message, [sender])

logs a normal event. Ignored if threshold > this.normalLevel (default: 10)

Kind: static method of `Logger`

Param	Type	Description
message	`string`	event to be logged
[sender]	`object`	*(optional)* sender of the message

Logger.debug(message, [sender])

logs debugging information. Ignored if threshold > this.debugLevel (default: 0)

Kind: static method of `Logger`

Param	Type	Description
message	`string`	event to be logged
[sender]	`object`	*(optional)* sender of the message

Logger.setThreshold(severity)

Sets threshold for logging

Kind: static method of `Logger`

Param	Type	Description
severity	`number`	a message has to have before being logged

Example

```
// By default, debug doesn't log
AnyBoard.debug("Hi")  // does not log
```

Example

```
// But you can lower the thresholdlevel
AnyBoard.Logger.setThreshold(AnyBoard.Logger.debugLevel)
AnyBoard.debug("I'm here afterall!")  // logs
```

Example

```
// Or increase it to avoid certain logging
AnyBoard.Logger.setThreshold(AnyBoard.Logger.errorLevel)
AnyBoard.warn("The tardis has arrived!")  // does not log
```

Example

```
// But you can never avoid errors
AnyBoard.Logger.setThreshold(AnyBoard.Logger.errorLevel+1)
AnyBoard.error("The Doctor is dead!!")  // logs
```

AnyBoard.Utils

Utility functions for AnyBoard

Kind: static property of `AnyBoard`

Utils.isEqual(a, b, [aStack], [bStack]) ⇒ `boolean`

Returns whether or not two objects are equal. Works with objects, dictionaries, and arrays as well.

Kind: static method of `Utils`
Returns: `boolean` - whether or not the items were equal

Param	Type	Description
a	`object` | `Array` | `String` | `number` | `boolean`	item to compare
b	`object` | `Array` | `String` | `number` | `boolean`	item to compare against a
[aStack]	`Array`	*(optional)* array of items to further compare
[bStack]	`Array`	*(optional)* array of items to further compare

Example

```
var tardis = {"quality": "awesome"}
var smardis = {"quality": "shabby"}
var drWhoCar = {"quality": "awesome"}

// Returns true
AnyBoard.Utils.isEqual(tardis, drWhoCar)

// Returns false
AnyBoard.Utils.isEqual(tardis, smardis)
```

playDrawCallback : `function`

This type of callback will be called when card is drawn or played

Kind: global typedef

Param	Type	Description
card	`Card`	that is played
player	`Player`	that played the card
[options]	`object`	*(optional)* custom options as extra parameter when AnyBoard.Player.play was called

simpleTriggerCallback : `function`

Type of callback called upon triggering of events

Kind: global typedef

Param	Type	Description

event	`string`	name of event
[options]	`object`	*(optional)* options called with the triggering of that event

stdStringCallback : `function`

Generic callback returning a string param

Kind: global typedef

Param	Type
string	`string`

stdBoolCallback : `function`

Generic callback returning a bool param

Kind: global typedef

Param	Type
boolean	`boolean`

stdNoParamCallback : `function`

Generic callback without params

Kind: global typedef

onScanCallback : `function`

Type of callback called upon detecting a token

Kind: global typedef

Param	Type	Description
token	`BaseToken`	discovered token

stdErrorCallback : `function`

This type of callback will be called upon failure to complete a function

Kind: global typedef

Param	Type
errorMessage	`string`