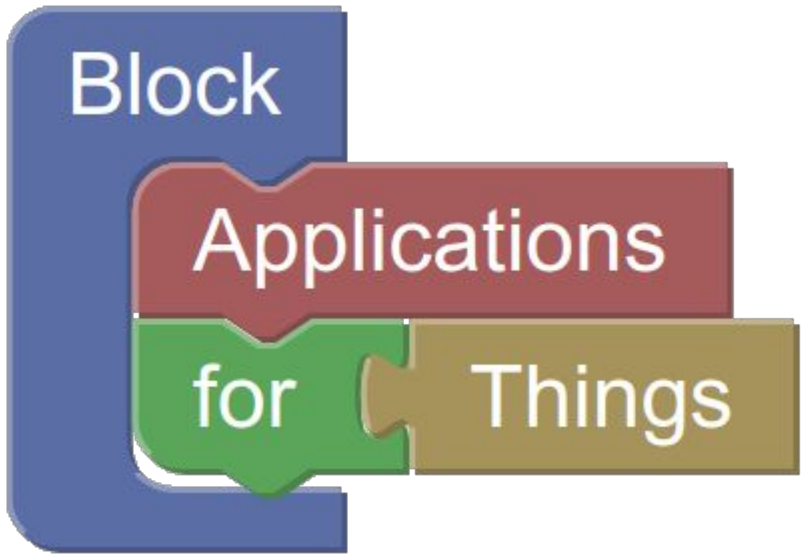
Friedrich-Alexander-Universität
Fachbereich Wirtschafts- und
Sozialwissenschaften | WiSo

Visual Programming with WoT

Thomas Wehr - 5.6.2025

Agenda



Motivation

We had a few Bluetooth and HID devices that we wanted to write programs for.

Requirements:

- Portable: Quick Installation, usable on Linux and Windows
- No programming skills necessary
- Add new devices without adapting anything
- Support for Bluetooth and HID devices

Motivation

We had a few Bluetooth and HID devices that we wanted to write programs for.

Requirements:

- Portable: Quick Installation, usable on Linux and Windows → Browser-based
- No program skills necessary
- Add new devices without adapting anything
- Support for Bluetooth and HID devices

Motivation

We had a few Bluetooth and HID devices that we wanted to write programs for.

Requirements:

- Portable: Quick Installation, usable on Linux and Windows → Browser-based
- No program skills necessary → Visual Programming
- Add new devices without adapting anything
- Support for Bluetooth and HID devices

Motivation

We had a few Bluetooth and HID devices that we wanted to write programs for.

Requirements:

– Portable: Quick Installation, usable on Linux and Windows	→ Browser-based
– No program skills necessary	→ Visual Programming
– Add new devices without adapting anything	→ Web of Things
– Support for Bluetooth and HID devices	

Motivation

We had a few Bluetooth and HID devices that we wanted to write programs for.

Requirements:

– Portable: Quick Installation, usable on Linux and Windows	→ Browser-based
– No program skills necessary	→ Visual Programming
– Add new devices without adapting anything	→ Web of Things
– Support for Bluetooth and HID devices	→ Custom Protocol Bindings

Visual Programming

Arranging graphical elements to create programs

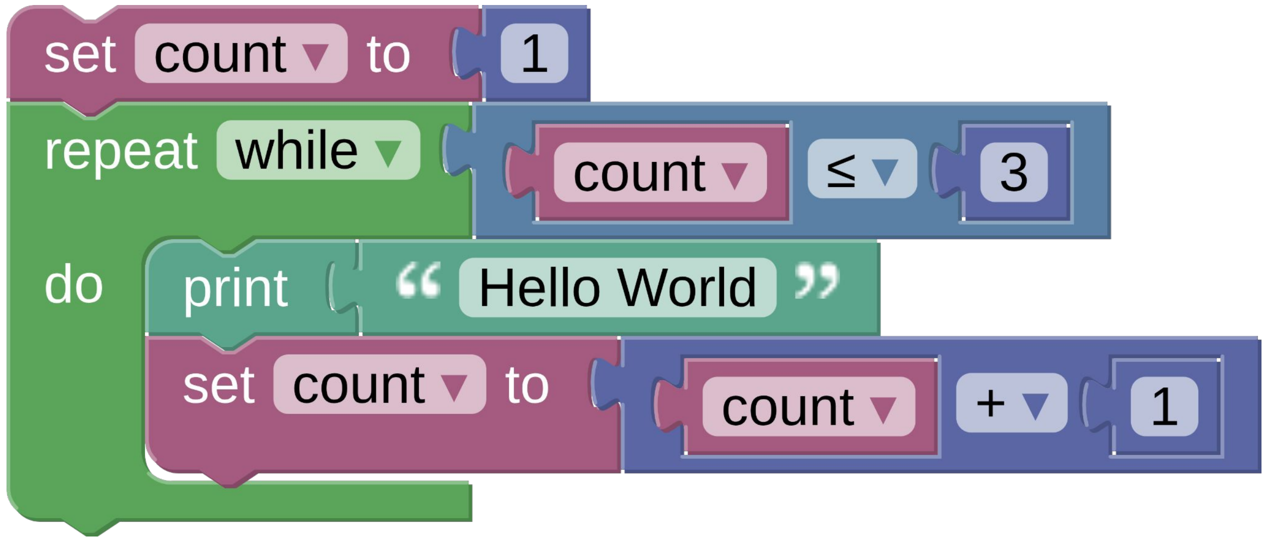
There are many different types of visual programming languages, such as flowcharts, node graphs, state machines, and dataflow programming.
We based our approach on *block-based programming*: Putting compatible “blocks” together like puzzle pieces.

Visual Programming

Arranging graphical elements to create programs

There are many different types of visual programming languages, such as flowcharts, node graphs, state machines, and dataflow programming.
We based our approach on *block-based programming*: Putting compatible “blocks” together like puzzle pieces.

- Often used for teaching programming to children



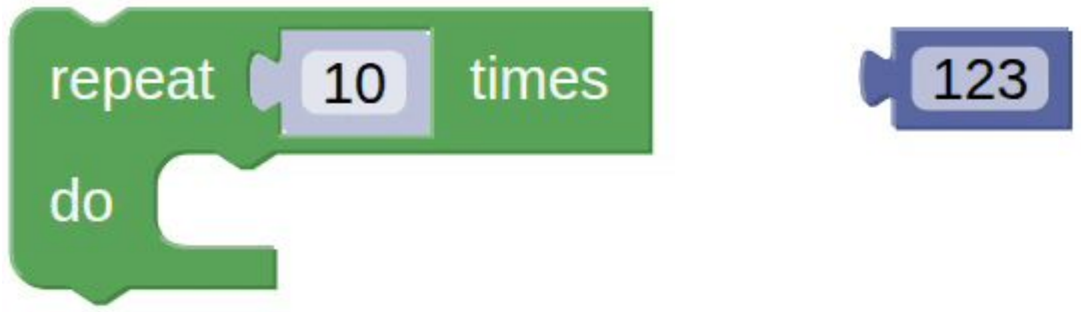
```
set count ▾ to 1
repeat while ▾ count ▾ ≤ ▾ 3
do
    print “ Hello World ”
    set count ▾ to count ▾ + ▾ 1
```

Visual Programming

Arranging graphical elements to create programs

There are many different types of visual programming languages, such as flowcharts, node graphs, state machines, and dataflow programming.
We based our approach on *block-based programming*: Putting compatible “blocks” together like puzzle pieces.

- Often used for teaching programming to children
- Compatible elements snap together

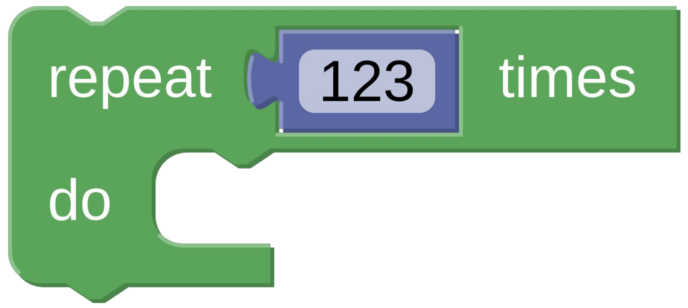


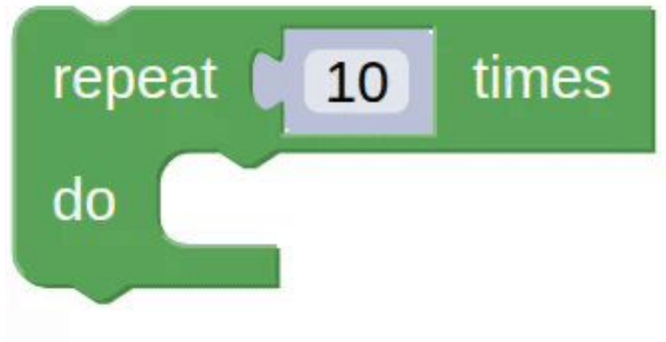
Visual Programming

Arranging graphical elements to create programs

There are many different types of visual programming languages, such as flowcharts, node graphs, state machines, and dataflow programming.
We based our approach on *block-based programming*: Putting compatible "blocks" together like puzzle pieces.

- Often used for teaching programming to children
- Compatible elements snap together
- No syntax errors possible as incompatible pieces won't snap together



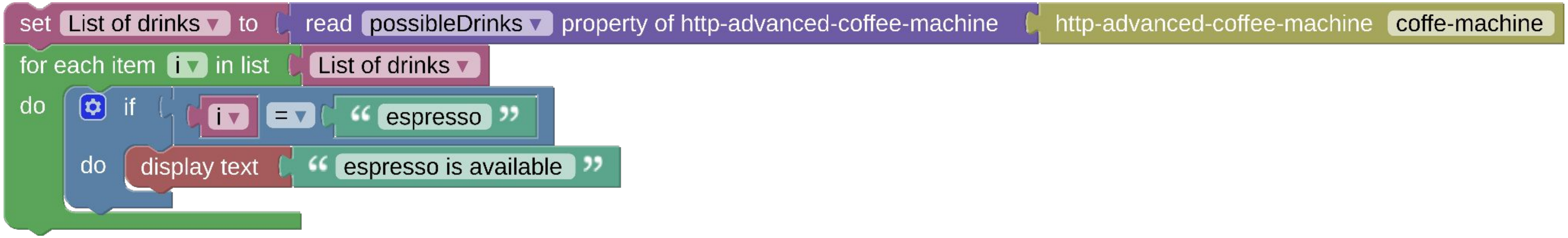


Visual Programming with the WoT

BLAST - Block Applications for Things (https://github.com/wintechis/blast)

is a browser-based visual programming environment for creating WoT Consumers using blocks.

- Blocks for control flow structures, like loops, if statements, blocks for iterating over lists, …



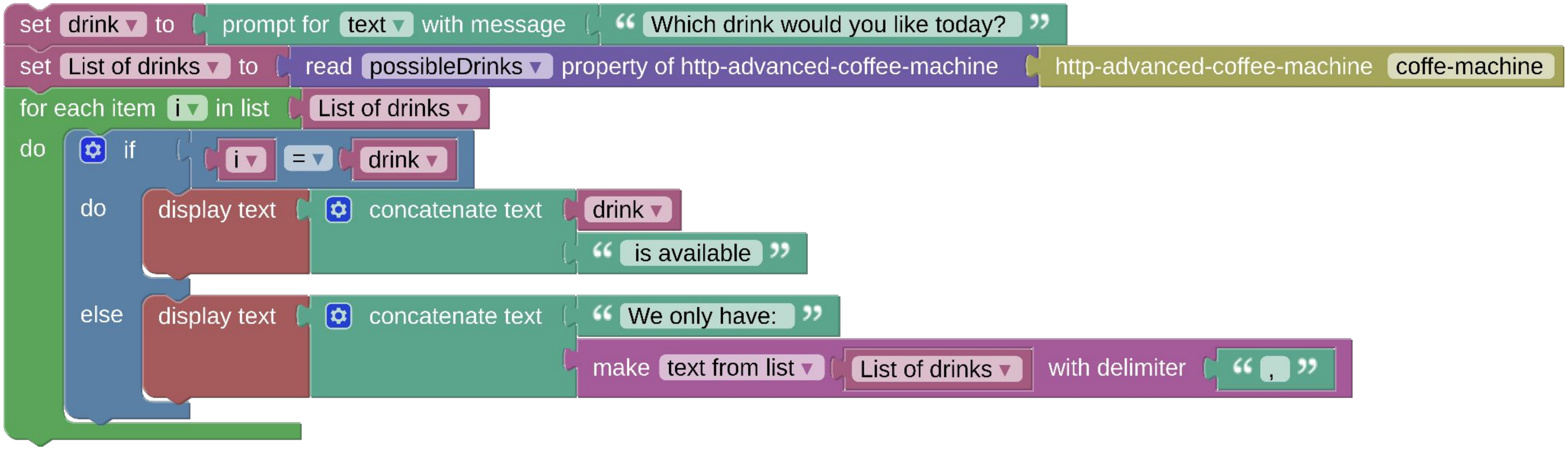
```
set List of drinks ▾ to  read possibleDrinks ▾ property of http-advanced-coffee-machine  http-advanced-coffee-machine coffe-machine
for each item i ▾ in list  List of drinks ▾
do  if  i ▾ = ▾ " espresso "
    do  display text " espresso is available "
```

Visual Programming with the WoT

BLAST - Block Applications for Things (https://github.com/wintechis/blast)

is a browser-based visual programming environment for creating WoT Consumers using blocks.

- Blocks for control flow structures, like loops, if statements, blocks for iterating over lists, …
- Text input and output, common string operations (concatenate, length, trim, substring, char at, to uppercase, …)



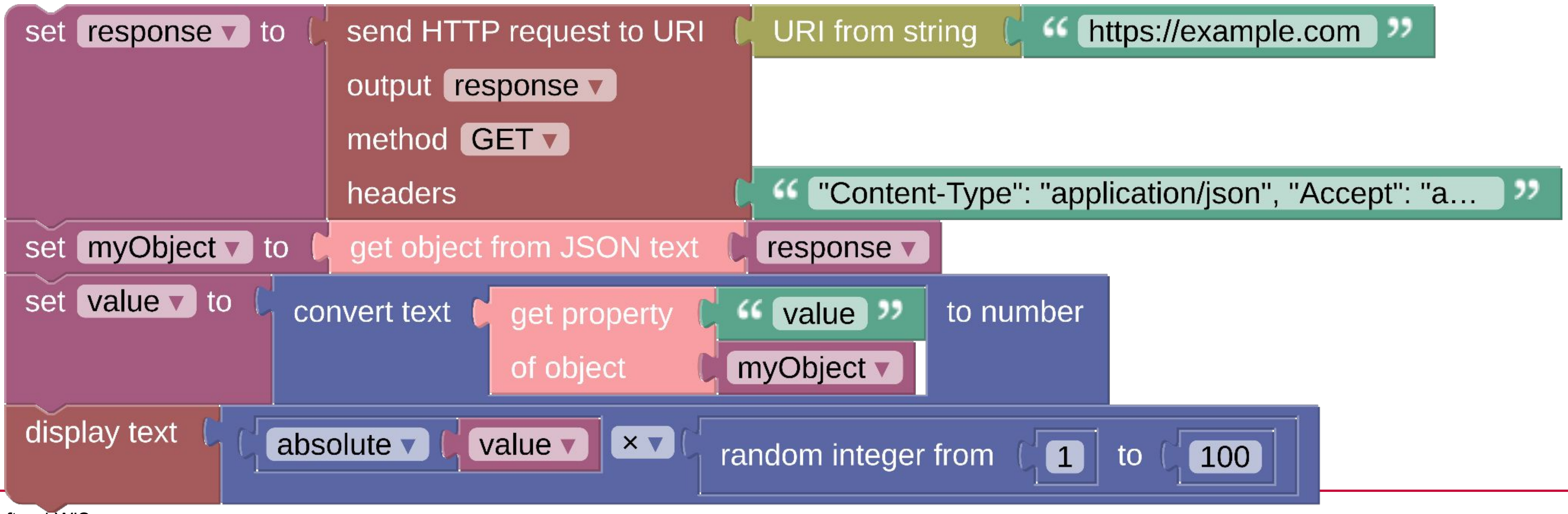
```
set drink ▾ to prompt for text ▾ with message " Which drink would you like today? "
set List of drinks ▾ to read possibleDrinks ▾ property of http-advanced-coffee-machine http-advanced-coffee-machine coffe-machine
for each item i ▾ in list List of drinks ▾
do  if  i ▾ = ▾ drink ▾
    do  display text  concatenate text  drink ▾
                                        " is available "
    else display text concatenate text  " We only have: "
                                        make text from list ▾ List of drinks ▾ with delimiter " , "
```

Visual Programming with the WoT

BLAST - Block Applications for Things (https://github.com/wintechis/blast)

is a browser-based visual programming environment for creating WoT Consumers using blocks.

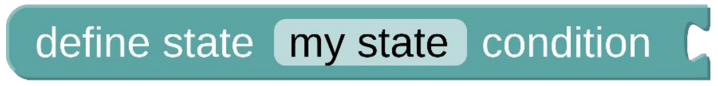
- Blocks for control flow structures, like loops, if statements, blocks for iterating over lists, …
- Text input and output, common string operations (concatenate, length, trim, substring, char at, to uppercase, …)
- Many more blocks for common programming constructs: operations on numbers, HTTP requests, text to speech / speech to text, functions, objects, …

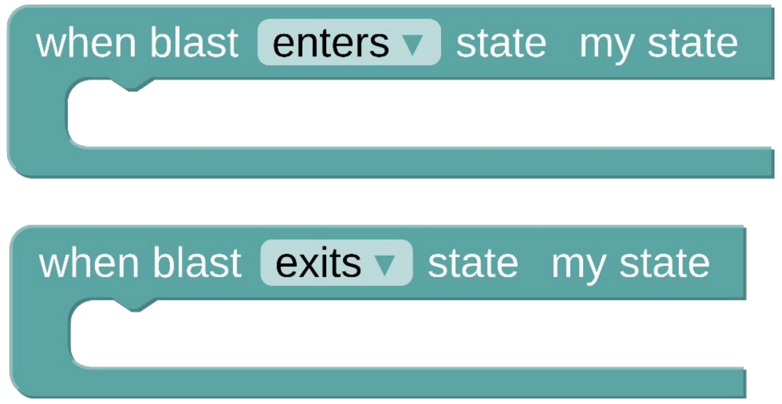


```
set response ▾ to  send HTTP request to URI  URI from string  " https://example.com "
                   output response ▾
                   method GET ▾
                   headers  " "Content-Type": "application/json", "Accept": "a… "
set myObject ▾ to  get object from JSON text  response ▾
set value ▾ to  convert text  get property  " value "  to number
                              of object     myObject ▾
display text  absolute ▾  value ▾  × ▾  random integer from 1 to 100
```

BLAST Event Loop

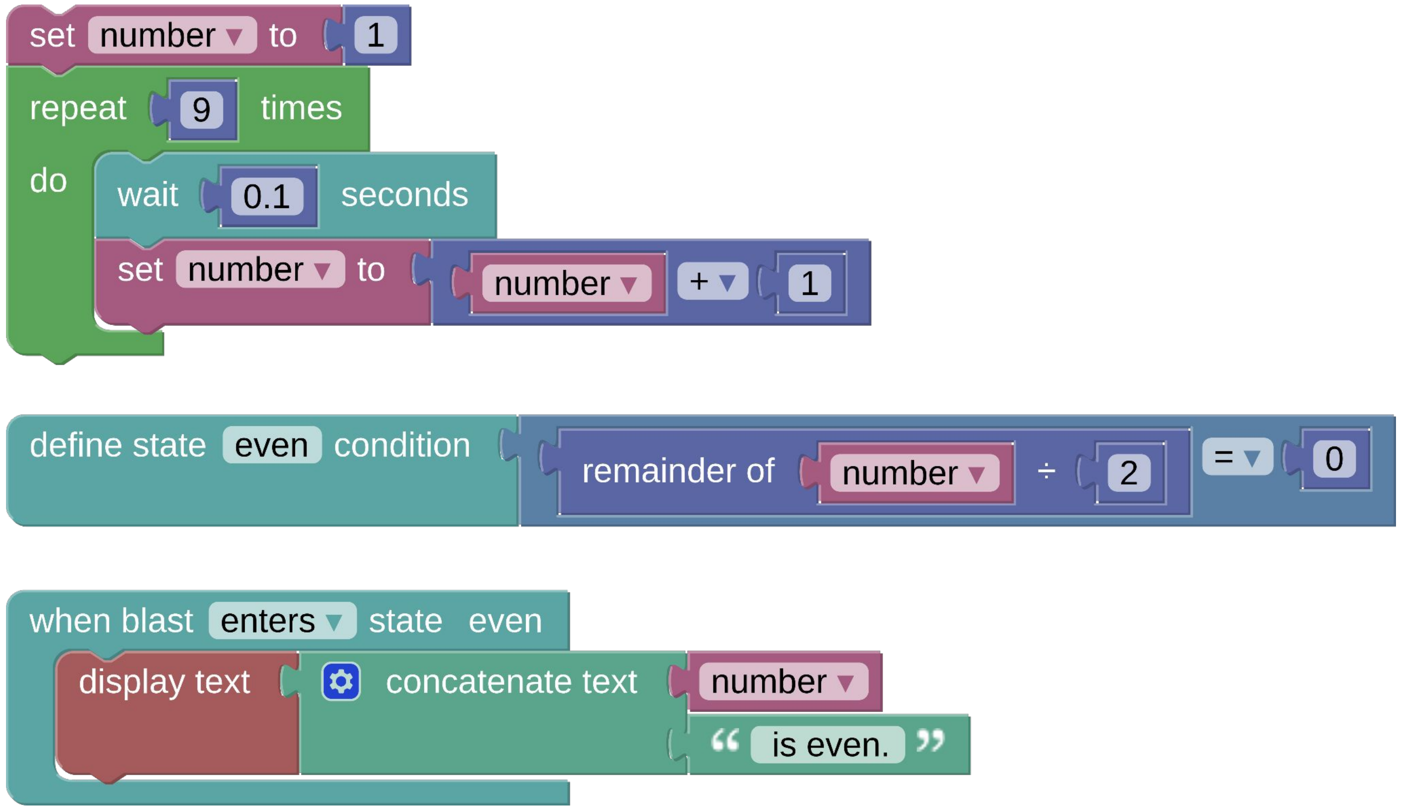
BLAST allows defining states and executing code on state transitions





BLAST Event Loop

BLAST allows defining states and executing code on state transitions

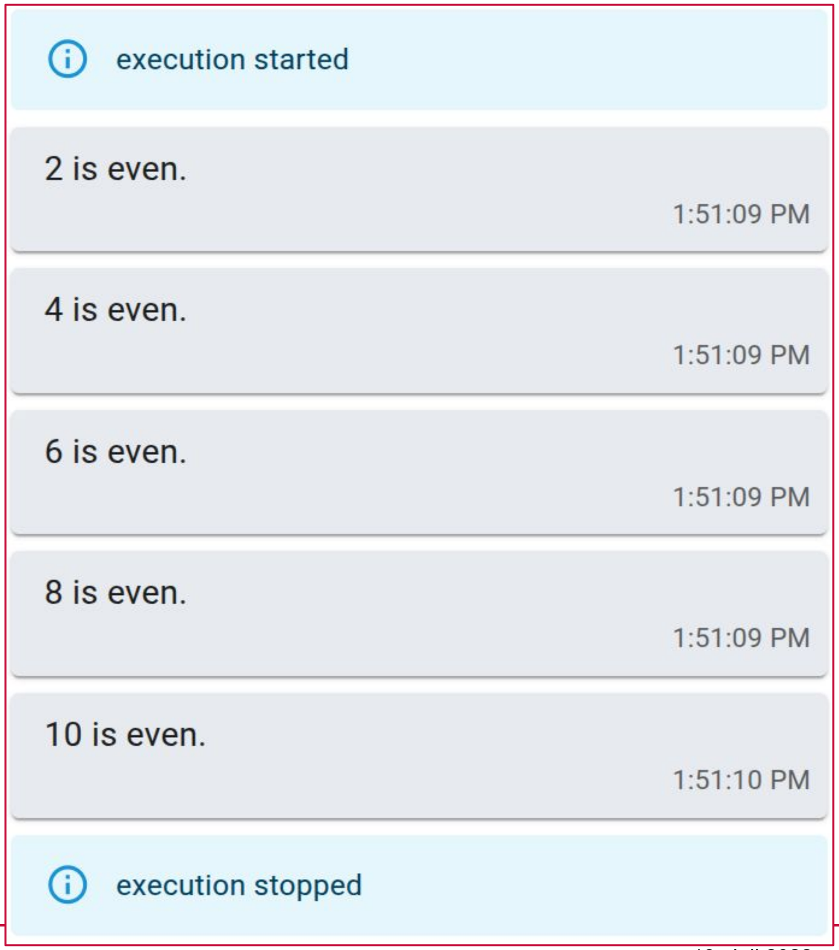


```
set number ▾ to 1
repeat 9 times
do  wait 0.1 seconds
    set number ▾ to number ▾ + ▾ 1

define state even condition remainder of number ▾ ÷ 2 = ▾ 0

when blast enters ▾ state even
    display text concatenate text number ▾
                                  " is even. "
```

output:



```
execution started

2 is even.
                1:51:09 PM
4 is even.
                1:51:09 PM
6 is even.
                1:51:09 PM
8 is even.
                1:51:09 PM
10 is even.
                1:51:10 PM

execution stopped
```

Generating Blocks from TDs

In addition to the pre-defined blocks, BLAST can consume TDs to generate blocks dynamically for interacting with additional devices, i.e.:

- reading and writing properties
- invoking actions
- subscribing to events

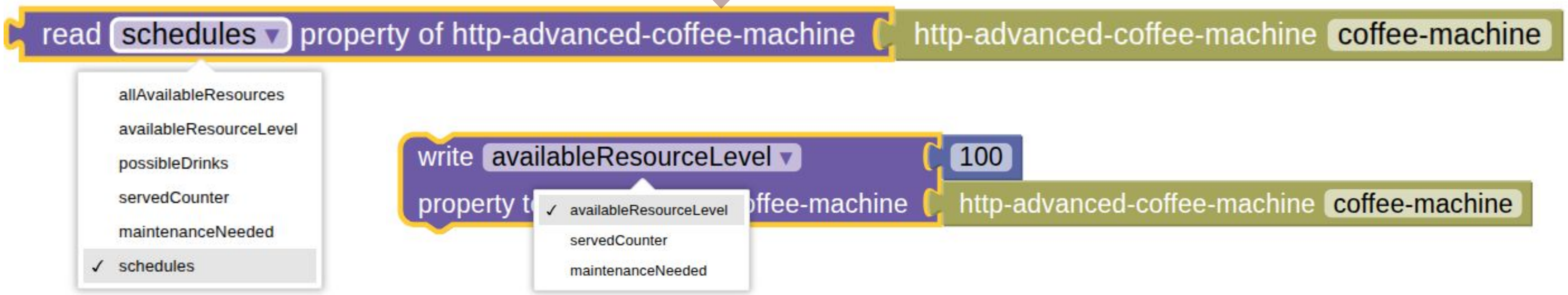
Generating Blocks from TDs

Reading and Writing Properties

```
"properties": {
   "allAvailableResources": { … },
   "availableResourceLevel": { … },
  "possibleDrinks": { … },
  "servedCounter": { … },
  "maintenanceNeeded": { … },
  "schedules": { … },
}
```

To generate property-blocks, BLAST parses the Property's:

- `type`, to make sure only blocks with the correct data type are attached
- `readOnly` and `writeOnly` values, to generate dropdown entries in the Property's read or write block
- `description` to display a hover text



read schedules ▾ property of http-advanced-coffee-machine
http-advanced-coffee-machine coffee-machine
allAvailableResources
availableResourceLevel
possibleDrinks
servedCounter
maintenanceNeeded
✓ schedules
write availableResourceLevel ▾
100
property t… …ffee-machine
✓ availableResourceLevel
servedCounter
maintenanceNeeded
http-advanced-coffee-machine coffee-machine

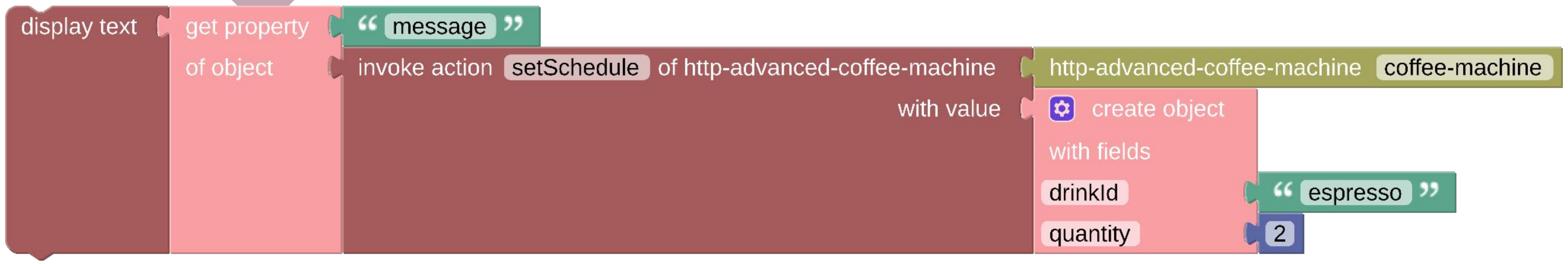
Generating Blocks from TDs

Invoking Actions

```
"actions": {
 "setSchedule": {
  "description": "...",
  "input": {
   "type": "object",
   "properties": {
    "drinkId": {
     "type": "string",
    },
    "quantity": {
     "type": "integer",
     "minimum": 1,
     "maximum": 5
    },
    ...
   },
   "output": {
    "type": "object",
    "properties": {
     "result": { "type": "boolean" },
     "message": { "type": "string" }
    }
   },
   "forms": [ ... ]
  }
 }
}
```

To generate action-blocks, BLAST parses the Action's:

- input and output `type`, to make sure only blocks with the correct data type are attached
- `description` to display a hover text



display text | get property “message” of object | invoke action setSchedule of http-advanced-coffee-machine | http-advanced-coffee-machine coffee-machine | with value | create object with fields | drinkId “espresso” | quantity 2

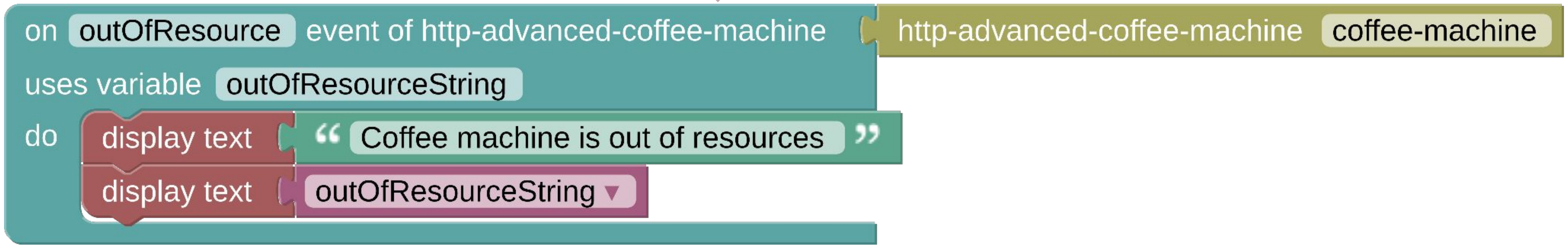
Generating Blocks from TDs

Subscribing to Events

```
"events": {
 "outOfResource": {
  "description": "...",
  "data": {
   "type": "string"
  },
  "forms": [ ... ]
 }
}
```

To generate event-blocks, BLAST parses the Event's:

- `data: {type}`, to make sure only blocks with the correct data type are attached
- `description` to display a hover text



```
on outOfResource event of http-advanced-coffee-machine   http-advanced-coffee-machine coffee-machine
uses variable outOfResourceString
do  display text “ Coffee machine is out of resources ”
    display text outOfResourceString ▾
```

node-wot Modifications

BLAST is based on node-wot but it also adds and extends some features.

- Additional protocol bindings for Bluetooth and HID devices, mapping the WoT operations to the *web-bluetooth* and *web-hid* APIs and to operations of the *ble-host* and *node-hid* packages.
- Extending the *octet-stream* codec with two properties `scale` and `default`

node-wot Modifications

scale example

Example for `scale` usage: RuuviTag

docs:

Offset	Allowed Values	Description
0	5	Data format (8bit)
1-2	-32767 … 32767	Temperature in 0.005 degrees
3-4	0 … 40 000	Humidity (16bit unsigned) **in 0.0025%** (0-163.83% range, though realistically 0-100%)
5-6	0 … 65534	Pressure (16bit unsigned) in 1 Pa units, with offset of -50 000 Pa
…		

TD:

```
"events": {
  "UART data": {
    "title": "Ruuvi Data data",
    "description": "…",
    "data": {
      "type": "object",
      "properties": {
        …
        "humidity": {
          "type": "number",
          "ex:bitOffset": 24,
          "ex:bitLength": 16,
          "signed": false,
          "scale": 0.0025,
          "unit": "qudtUnit:PERCENT"
        },
        …
 } } } }
```

node-wot Modifications

default example

Example for `default` usage: BLE LED Controller

docs:

To set the LED color send the byte string **`7e070503xxxxxx00ef`**, where:

byte	description
7e	command start
07	command length
05	command id
03	command sub-id
xx	command arg1 (red value)
xx	command arg2 (green value)
xx	command arg3 (blue value)
00	not used
ef	command end

node-wot Modifications

default example

Example for `default` usage: BLE LED Controller

docs:

To set the LED color send the byte string **`7e07050`**

byte	description
7e	command start
07	command length
05	command id
03	command sub-id
xx	command arg1 (red value)
xx	command arg2 (green value)
xx	command arg3 (blue value)
00	not used
ef	command end

```
"properties": {
    "colour": {
      ...
      "type": "object",
      "properties": {
        "commandStart":{
          "type": "integer",
          "default": 126,
          "description": "Command start byte.",
          "ex:bitOffset": 0,
          "ex:bitLength": 8
        },
        "commandLength": {
          "type": "integer",
          "default": 7,
          "description": "Command length.",
          "ex:bitOffset": 8,
          "ex:bitLength": 8
        },
        "commandId": {
          "type": "integer",
          "default": 5,
          "description": "Command type.",
          "ex:bitOffset": 16,
          "ex:bitLength": 8
        },
        "commandSubId": {
          "type": "integer",
          "default": 3,
          "description": "Command sub type.",
          "ex:bitOffset": 24,
          "ex:bitLength": 8
        },
        "R": {
          "type": "integer",
```

node-wot Modifications

default example

Example for `default` usage: BLE LED Controller

docs:

To set the LED color send the byte string **`7e07050`**

byte	description
7e	**command start**
07	**command length**
05	**command id**
03	**command sub-id**
xx	command arg1 (red value)
xx	command arg2 (green value)
xx	command arg3 (blue value)
00	not used
ef	command end

```
"properties": {
    "colour": {
      ...
      "type": "object",
      "properties": {
        "commandStart":{
          "type": "integer",
          "default": 126,
          "description": "Command start byte.",
          "ex:bitOffset": 0,
          "ex:bitLength": 8
        },
        "commandLength": {
          "type": "integer",
          "default": 7,
          "description": "Command length.",
          "ex:bitOffset": 8,
          "ex:bitLength": 8
        },
        "commandId": {
          "type": "integer",
          "default": 5,
          "description": "Command type.",
          "ex:bitOffset": 16,
          "ex:bitLength": 8
        },
        "commandSubId": {
          "type": "integer",
          "default": 3,
          "description": "Command sub type.",
          "ex:bitOffset": 24,
          "ex:bitLength": 8
        },
        "R": {
          "type": "integer",
```

node-wot Modifications

default example

Example for `default` usage: BLE LED Controller

docs:

To set the LED color send the byte string **`7e07050`**

byte	description
7e	command start
07	command length
05	command id
03	command sub-id
xx	**command arg1 (red value)**
xx	**command arg2 (green value)**
xx	**command arg3 (blue value)**
00	not used
ef	command end

```
"properties": {
    "colour": {
      ...
      "type": "object",
      "properties": {
        ...
        "R": {
          "type": "integer",
          "minimum": 0,
          "maximum": 255,
          "description": "Red value.",
          "ex:bitOffset": 32,
          "ex:bitLength": 8
        },
        "G": {
          "type": "integer",
          "minimum": 0,
          "maximum": 255,
          "description": "Green value.",
          "ex:bitOffset": 40,
          "ex:bitLength": 8
        },
        "B": {
          "type": "integer",
          "minimum": 0,
          "maximum": 255,
          "description": "Blue value.",
          "ex:bitOffset": 48,
          "ex:bitLength": 8
        },
        "unknown": {
          "type": "integer",
          "default": 0,
          "ex:bitOffset": 56,
          "ex:bitLength": 8
```

node-wot Modifications

default example

Example for `default` usage: BLE LED Controller

docs:

To set the LED color send the byte string **`7e07050`**

byte	description
7e	command start
07	command length
05	command id
03	command sub-id
xx	command arg1 (red value)
xx	command arg2 (green value)
xx	command arg3 (blue value)
00	**not used**
ef	**command end**

```
"properties": {
    "colour": {
      ...
      "type": "object",
      "properties": {
        ...
        "unknown": {
          "type": "integer",
          "default": 0,
          "ex:bitOffset": 56,
          "ex:bitLength": 8
        },
        "commandEnd": {
          "type": "integer",
          "default": 239,
          "description": "Command end byte.",
          "ex:bitOffset": 64,
          "ex:bitLength": 8
        }
      },
      "required": ["R","G","B"],
      "forms": [
        {
          "op": "writeproperty",
          "href": "./0000fff0-0000-1000-8000-00805f9b34fb/0000fff3-0000-1000-8000-00805f9b34fb",
          "contentType": "application/octet-stream;length=9;signed=false"
        }
      ]
     },
```

node-wot Modifications

default example

Now instead of

```
ledController.writeProperty('colour', {
  commandStart: 126,
  commandLength: 7,
  commandId: 5,
  commandSubId: 3,
  R: 255,
  G: 0,
  B: 0,
  unknown: 0,
  commandEnd: 239
});
```

we can write

```
ledController.writeProperty('colour', {R: 255, G: 0, B: 0});
```

Demo

Conclusion

BLAST - Block Applications for Things is a visual programming and execution environment for WoT Consumers.

- Many pre-defined blocks for basic programming constructs
- Pre-defined blocks for some devices
- Can consume TDs to dynamically add new devices
- Event loop for defining states and events
- Protocol Bindings for interacting with Bluetooth and HID devices
- Extends octet-stream codec to support `scale` and `default` keywords

https://github.com/wintechis/blast

Thank you for your attention

Contact Details:

Thomas Wehr
Research Assistant

Chair of Technical Information Systems
Friedrich-Alexander-Universität Erlangen-Nürnberg

E-Mail: thomas.wehr@fau.de