



Norwegian University of
Science and Technology

Extending the TILES Toolkit - from Ideation to Prototyping

Anders Riise Mæhlum

Master of Science in Computer Science

Submission date: June 2017

Supervisor: Monica Divitini, IDI

Co-supervisor: Simone Mora, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Internet of Things is a term that is slowly working its way into people's vocabulary. The steadily decline in hardware prices is making sensors and actuators more accessible, and embedded hardware devices can be acquired for almost nothing. The TILES toolkit is a hardware and software platform that enables IoT application prototyping, using the custom TILES Squares as hardware devices for interaction with the real world. The TILES toolkit provides several layers of abstractions by hiding the communication and hardware complexities from the application developers. The TILES toolkit has successfully established a fundamental infrastructure with which non-experts and experts alike can prototype their IoT applications. In addition, the TILES Card ideation process has been developed to enable non-experts to express their creativity by iteratively expand their own application ideas. The TILES toolkit, however, has failed to provide tools enabling non-experts to transition from ideation to prototyping without help and guidance from toolkit experts. At the same time, the toolkit provides no means for the expert users to extend and customize the toolkit for their special application scenarios.

The research questions defined for this project are twofold. First, they put focus on the transition from the TILES Card ideation phase to application prototyping by non-experts, and their ability to rapidly prototype their application ideas. Secondly, they address the needs of expert users in customizing and extending the system with additional interfaces and hardware capabilities for creating a tailored toolkit for special usage areas. These questions are answered in this report through requirement analysis, design and creation to extend the TILES toolkit and evaluation of the extended system together with reviewing literature to assert the innovativeness of the work performed.

During this project, two new processes have been defined for supporting both non-experts and expert users in their TILES project. The requirement specification for the extended TILES toolkit resulted in an extensive modification of the implementation of the TILES Cloud and a new and improved web portal for the toolkit. As evaluation, two workshops with non-experts and a focus group with expert users were conducted. These evaluations have provided important insights into the needs and usage patterns of both types of users, and served as an evaluation and verification of the requirements and design developed during this research project. The new tools have been added to the TILES toolkit and will enable users to use the TILES toolkit for prototyping IoT applications and extend the toolkit for a variety of application domains.

Sammendrag

Tingenes internett (IoT) er en term som sakte men sikkert er i ferd med å bli en del av vårt vokabular. De stadig synkende prisene på maskinvare og sensorer gjør at vi i dag kan anskaffe dedikerte datamoduler for nesten ingenting. *TILES toolkit* eller *TILES verktøykasse* er en kombinert maskinvare- og programvareplattform som gjør det mulig å utvikle IoT prototypapplikasjoner ved å bruke *TILES Square* som maskinvare for å detektere hendelser i den virkelige verden. *TILES verktøykasse* består av flere lag med abstraksjoner som skjuler kompleksiteten med kommunikasjonsprotokoller og maskinvare for utviklere. En grunnleggende infrastruktur, som har muliggjort utvikling av IoT-applikasjoner utført av utviklere både med og uten erfaring, har blitt implementert i verktøykassen. I tillegg har *TILES Cards*, et kortspill, blitt utviklet som et verktøy som gjør at selv brukere uten kunnskap og erfaring med IoT kan uttrykke sin kreativitet ved å systematisk bruke kortene til å definere IoT-applikasjoner. På den annen side så finnes det svakheter med *TILES toolkit*. Overgangen fra idemyldring med *TILES Cards* til utvikling av prototyper med verktøykassen er fremdeles komplisert og krever veiledning fra TILES-eksperter. I tillegg finnes det ingen mulighet for erfarne brukere å utvide og skreddersy systemet til å brukes i mer spesifikke applikasjonsdomener.

Forskningsspørsmålene utarbeidet i dette prosjektet er todelt. For det første legger de vekt på overgangen fra ideutviklingsfasen til prototypfasen utført av ikke-eksperter og deres evne til å raskt lage prototyper av sine idéer. For det andre så adresserer de erfarne brukeres behov med å legge til ny funksjonalitet i maskin-, og programvare for å bruke TILES i spesifikke bruksområder. Disse spørsmålene vil besvares i denne oppgaven gjennom analyse av kravspesifikasjon, design og utvikling for å utvide verktøykassen, evaluering av det utvidede systemet og analyse av relevant litteratur.

I dette prosjektet har to nye prosesser for å støtte både erfarne og ikke-erfarne brukere i deres bruk av systemet blitt introdusert. Kravspesifikasjonen av det nye systemet har medført en omfattende endring av systemet og en ny og bedre nettportal har blitt utarbeidet. Som evaluering har to brukerstudier blitt gjennomført for ikke-eksperter og en fokusgruppe med ekspertbrukere har blitt organisert. Disse to rundene med evaluering har gitt et viktig innblikk i behov og brukermønstre til både erfarne og ikke-erfarne brukere, og har hjulpet til å verifisere kravspesifikasjonen av det nye og forbedrede systemet. Den nye og forbedrede *TILES verktøykassen* vil gjøre det mulig å utvikle IoT prototypapplikasjoner, samt utvide verktøykassen til spesifikke formål.

Contents

Abstract	I
Sammendrag	III
Contents	V
List of Figures	XI
List of Tables	XIII
List of Code snippets	XV
Abbreviations	XVII
1 Introduction	1
1.1 Motivation.....	1
1.2 Context.....	2
1.3 Problem elaboration	2
1.4 Research Questions	3
1.5 Research Method.....	3
1.6 Results	5
1.7 Outline.....	6
2 TILES toolkit	7
2.1 Ideation phase	7
2.2 Prototyping phase.....	9
2.2.1 Toolkit architecture.....	9
3 From Ideation to Prototyping	15
3.1 TILES toolkit Processes.....	15
3.1.1 TILES toolkit Application Development Process.....	16

3.1.2	TILES toolkit Extension Process.....	19
3.2	Toolkit Documentation Section	20
3.2.1	Outline	21
4	Requirement Specification	25
4.1	Application appropriation	26
4.2	Rule Engine Development Environment	28
4.3	TILES toolkit IDE.....	29
4.4	TILES Cloud web portal	30
5	Design	33
5.1	Users	33
5.2	TILES toolkit revised.....	34
5.2.1	Requirement allocation	35
5.3	TILES Cloud web portal	36
6	Implementation	39
6.1	Application Appropriation	39
6.1.1	TilesAscoltatore	39
6.1.2	TilesApi.....	41
6.1.3	MongoDB	41
6.1.4	Requirement Rationale.....	42
6.2	Rule Engine Environment.....	46
6.2.1	MongoDB	47
6.2.2	REST API	48
6.2.3	Requirement Rationale.....	49
6.3	TILES toolkit IDE.....	51
6.3.1	Cloud9.....	51
6.3.2	Background processes	52
6.3.3	Requirement Rationale.....	53
7	Evaluation	57
7.1	Non-expert workshops	57
7.1.1	Objectives	57
7.1.2	Research tools.....	58
7.1.3	Participants	60
7.1.4	Setting up and running the workshop	60
7.1.5	Results.....	63
7.2	Expert focus group.....	70
7.2.1	Objectives	70
7.2.2	Research tools.....	70

7.2.3	Participants	71
7.2.4	Setting up and running the focus group	71
7.2.5	Results	71
7.3	Summary	73
8	Related Work	75
8.1	Commercial toolkits	75
8.1.1	End-to-end.....	76
8.1.2	Partial end-to-end	77
8.1.3	Not end-to-end.....	77
8.2	Literature	78
8.2.1	BitWear	80
8.2.2	Bloctopus.....	80
8.2.3	BRIX	81
8.2.4	Functionality vs. Ease of use	82
9	Conclusions	83
9.1	Summary of results	83
9.1.1	RQ1: How to support rapid prototyping and deployment of IoT applications using TILES?.....	83
9.1.2	RQ1.1: How to support rapid prototyping of ideas created with TILES Cards?.....	83
9.1.3	RQ1.2: How to support TILES application development by non-experts?.....	84
9.1.4	RQ1.2: How to support toolkit extension for expert users with minimal efforts without breaking development support for non-experts?	84
9.2	Discussion	85
9.3	Future work	86
9.3.1	Bring TILES to the classroom	86
9.3.2	Bring TILES to the market.....	87
9.3.3	Study <i>In-the-wild</i>	87
9.3.4	Toolkit improvements	87
	References	91
	Appendix A	95
A	Deploying TILES Cloud with Cloud9 as TIDE	95
A.1	About DigitalOcean	95
A.2	Setting up droplets in DigitalOcean	95
A.2.1	Install Node.js and npm on Ubuntu via terminal	96
A.2.2	Cloning TILES Cloud git repository	96
A.2.3	Installing MongoDB	96

A.2.4	Running Node.js server as a service using PM2.....	97
A.3	Installing Cloud9 core.....	98

Appendix B **101**

B TILES toolkit Documentation Section **101**

B.1	Getting Started	101
	Introduction to the documentation	101
	Getting Started	102
	TILES toolkit	103
	Development Environments.....	104
	TILES Square Primitives.....	107
B.2	App. Development Process.....	108
	0. Process Description	108
	1. Ideation Phase	110
	2. Create User	111
	3. Create Application	111
	4. List Physical Objects.....	112
	5. Launch Dev. Env.....	114
	6. Code Application.....	114
	7. Test Application	115
	8. Iterate step 6-7	118
B.3	JavaScript API	118
	1. Introduction	118
	2. EventReader API	120
	3. Map HUMAN ACTIONS	123
	4. Map FEEDBACK	124
	5. Map SERVICES.....	126
	6. Example Scenario 1.....	128
	7. Example Scenario 2.....	130
B.4	Toolkit Extension Process.....	132
	0. Process Description	132
	1. Device Development.....	133
	2. Library Development	136
	3. API Deployment.....	138
	4. TILES Card Deck Creation.....	139
B.5	Rule Engine API	139
	1. Introduction	139
	2. Map HUMAN ACTIONS	141
	3. Map FEEDBACK	141

4. MAP SERVICES.....	142
5. Example scenario	144

List of Figures

Figure 1-1, Design Science Research, Knowledge Base (adopted from [9]).....	4
Figure 1-2, Design Science Research, Environment (adopted from [9])	4
Figure 1-3, Model of The Research Process (adapted from [25]).....	5
Figure 2-1, TILES Idea Generator Board Credit: Simone Mora.....	8
Figure 2-2, TILES Cards Credit: Simone Mora.....	9
Figure 2-3, TILES architecture overview with “user bar” [23]	10
Figure 2-4, (a) RFDuino shields, (b) Connected RFDuino shields forming first TILES Square prototype	11
Figure 2-5, TILES Square, RFDuino, embedded hardware [20]	11
Figure 3-1, TILES toolkit Application Development Process	16
Figure 3-2, TADP object representation flow	18
Figure 3-3, TILES toolkit Extension Process.....	20
Figure 4-1, Old TILES Cloud web portal UI (a) All users, (b) Specific user with TILES Squares, (c) Specific TILES Square.....	30
Figure 5-1, TILES toolkit Use Case Diagram.....	34
Figure 5-2, Extended TILES toolkit system architecture.....	35
Figure 5-3, Toolkit architectural element requirement allocation.....	36
Figure 5-4, TILES Cloud web portal page graph.....	37
Figure 5-5, TILES Cloud web portal main navigation panel.....	37
Figure 7-1, Workshop application storyboard.....	61
Figure 7-2, Workshop questionnaire answers, regarding the documentation	66
Figure 7-3, Workshop questionnaire answers, regarding the prototyping phase	67
Figure 7-4, Workshop questionnaire answers, regarding the scenarios	68
Figure 7-5, Questionnaire answers, workshop 1, regarding the tools	68
Figure 7-6, Questionnaire answers, workshop 1, general	69

Figure A-1, DigitalOcean droplet configuration	96
Figure B-1, What you need for IoT prototyping with the TILES toolkit.....	102
Figure B-2, Layered architecture of the TILES toolkit.....	103
Figure B-3, Start hosting of the cloud9 web IDE for the application	104
Figure B-4, Cloud9 web IDE configured with TILES JavaScript API.....	105
Figure B-5, Rule Engine Development Environment	107
Figure B-6, RapIoT toolkit example primitives.....	108
Figure B-7, TILES toolkit Application Development Process (TADP)	110
Figure B-8, TILES Cards	110
Figure B-9, Creating a new user account	111
Figure B-10, List of Applications in TILES Cloud web portal.....	112
Figure B-11, Add application form in TILES Cloud web portal	112
Figure B-12, THINGS Cards from TILES Cards	113
Figure B-13, Application details, with two items	113
Figure B-14, Cloud Development Environment	114
Figure B-15, Cloud Development Environment	116
Figure B-16, Log into app with server address, username and port number	116
Figure B-17, List of available applications in gateway.....	117
Figure B-18, List of items in application	117
Figure B-19, Tap Watch.....	123
Figure B-20, Tilt Headgear	124
Figure B-21, Change color on Plant.....	125
Figure B-22, Vibrate Refrigerator.....	125
Figure B-23, Tilt Watch to send Email (using IFTTT).....	128
Figure B-24, TILES toolkit Extension Process.....	133
Figure B-25, Garage Control System application using Rule Engine Dev. Env.	140
Figure B-26, Add new TILE rule	141
Figure B-27, TILE rule form.....	141
Figure B-28, TILE rule form divided into HUMAN ACTIONS and FEEDBACK	141
Figure B-29, TILE rule form divided into HUMAN ACTIONS and FEEDBACK	142
Figure B-30, Defined TILE rules with HUMAN ACTIONS and FEEDBACK	142
Figure B-31, Add IFTTT personal key to Rule Engine application	142
Figure B-32, IFTTT rule form1	143
Figure B-33, Defined IFTTT rules with HUMAN ACTIONS and SERVICE.....	143
Figure B-34, IFTTT rule form2	143
Figure B-35, Defined IFTTT rules with SERVICE and FEEDBACK	143
Figure B-36, Rule Engine example program definition.....	145

List of Tables

Table 2-1, TILES Square interaction primitives [23]	12
Table 3-1, TILES toolkit Documentation Section outline	23
Table 4-1, System requirements [23]	26
Table 4-2, Application appropriation requirements.	27
Table 4-3, Rule Engine Requirements.	28
Table 4-4, TILES toolkit IDE requirements.....	29
Table 4-5, URS of TILES Cloud web portal.....	31
Table 6-1, AR1 implementation files	42
Table 6-2, AR2 implementation files	43
Table 6-3, AR3 implementation files	43
Table 6-4, AR4 implementation files	44
Table 6-5, AR5 implementation files	44
Table 6-6, AR6 implementation files	44
Table 6-7, AR8 implementation files	45
Table 6-8, AR9 implementation files	45
Table 6-9, AR10 implementation files	45
Table 6-10, AR11 implementation files	46
Table 6-11, RR1 implementation files	49
Table 6-12, RR2 implementation files	50
Table 6-13, RR3 implementation files	50
Table 6-14, RR5 implementation files	51
Table 6-15, IR3 implementation files	54
Table 6-16, IR3 implementation files	54
Table 6-17, IR4 implementation files	55
Table 6-18, IR5 implementation files	55
Table 6-19, IR7 implementation files	56
Table 7-1, Non-expert survey questions	59
Table 7-2, Non-expert workshop protocol	62

Table 8-1, Commercial toolkits for IoT prototyping [23]. Credit: Simone Mora.....	76
Table 8-2, Related literature for IoT and prototyping toolkits Credit: Simone Mora	79
Table B-1, TILES Square interaction primitives	108
Table B-2, Steps of TADP	109
Table B-3, Steps of TEP.....	133

List of Code snippets

Code snippet 6-1, TilesAscoltatore application topic subscription	40
Code snippet 6-2, TilesAscoltatore prototype unsubscribe	41
Code snippet 6-3, VirtualTiles MongoDB databasemodel	42
Code snippet 6-4, Applications MongoDB databasemodel	42
Code snippet 6-5, Tilehooks, MongoDB database model.....	47
Code snippet 6-6, Ifttthooks, MongoDB database model	48
Code snippet 6-7, Primitives, MongoDB database model	48
Code snippet 6-8, REST API, POST service, Create new Tilehook.....	49
Code snippet 6-9, Create Workspace helper function.....	53
Code snippet 7-1, First workshop, application code group A	64
Code snippet 7-2, First workshop, application code group B	65
Code snippet 7-3, EventReader extension for LED strip, group A.....	72
Code snippet 7-4, EventReader extension for LED strip, group B	73
Code snippet A-1, Installing Node.js and npm in Ubuntu VM.....	96
Code snippet A-2, Cloning git TILES repository in Ubuntu VM.....	96
Code snippet A-3, Installing MongoDB in Ubuntu VM	97
Code snippet A-4, Edit mongod.service in Ubuntu VM	97
Code snippet A-5, mongod.service configuration file in Ubuntu VM.....	97
Code snippet A-6, Starting MongoDB as a service	97
Code snippet A-7, Starting server with PM2	97
Code snippet A-8, Installing prerequisites for Cloud9 core.....	98
Code snippet A-9, Cloning Cloud9 core and install.....	98
Code snippet A-10, apache2 configuration for Cloud9.....	98
Code snippet A-11, Starting an instance of Cloud9 server (TIDE)	98
Code snippet B-1, tiles.js template file explained	119
Code snippet B-2, Initializing EventReader API	120
Code snippet B-3, EventReader API example	121
Code snippet B-4, Tile methods example	122
Code snippet B-5, TileClient example	123

Code snippet B-6, Tap Watch example.....	123
Code snippet B-7, Tilt Headgear example	124
Code snippet B-8, Change color on Plant example.....	125
Code snippet B-9, Vibrate Refrigerator example.....	125
Code snippet B-10, Initialize PostmanClient API.....	126
Code snippet B-11, PostmanClient example.....	126
Code snippet B-12, Initialize IFTTTClient API	127
Code snippet B-13, IFTTTClient example.....	127
Code snippet B-14, Tilt Watch to send Email example	128
Code snippet B-15, JavaScript example scenario 1	130
Code snippet B-16, JavaScript example scenario 2	132
Code snippet B-17, Firmware code, set output primitive LED to specific color.....	135
Code snippet B-18, read double tap in EventReader API.....	137
Code snippet B-19, getTile in EventReader API	137
Code snippet B-20, Placeholders of template files	138

Abbreviations

API	Application Programming Interface
BLE	Bluetooth Low Energy
CRUD	Create, Read, Update, Delete
CSS	Cascading Styling Sheet
DIY	Do It Yourself
DSL	Domain Specific Language
GUI	Graphical User Interface
HTML	Hypertext Modeling Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
IDE	Integrated Development Environment
IP	Internet Protocol
IoT	Internet of Things
JS	JavaScript
JSON	JavaScript Object Notation
LED	Light Emitting Diode
MQTT	Message Queuing Telemetry Transport
NTNU	Norwegian University of Science and Technology
OS	Operating system
REST	Representational State Transfer
RGB	Red Green Blue
RQ	Research Question
SDK	Software Development Kit
TADP	TILES toolkit Application Development Process
TCP	Transmission Control Protocol

TDS TILES toolkit Documentation Section

TEP TILES toolkit Extension Process

TIDE TILES toolkit Integrated Development Environment

UI User Interface

URL Uniform Resource Locator

UX User Experience

1 Introduction

The TILES toolkit¹ is a software and hardware framework for designing and implementing Internet of Things applications. The toolkit is composed of *design tools* supporting brainstorming on how to use IoT technology in everyday activities, *hardware components* for detecting physical events in the real world, *gateway and middleware* to translate the physical triggers into a digital representation of detected physical events, *development tools* for developing IoT applications and *development support tools* to enable people with little programming experience to develop fully functional IoT applications. All throughout the TILES Project, the needs and abilities of non-experts in prototyping IoT applications for real object augmentation has had an essential role. The previously concluded specialization project [23] laid the foundation of this thesis project with an introduction to, and thorough evaluation of, the TILES toolkit through conducting several user studies organized for non-experts. This thesis project will be devoted to extend the TILES toolkit and implement new components based on the results and discoveries of the specialization project in order to build a more robust toolkit with a higher degree of customization, in addition to enable the toolkit to be self-supporting by providing the necessary tools to be used without expert users.

1.1 Motivation

Prototyping Internet of Things systems is a complex process composed of many complicated procedures. It requires knowledge of embedded hardware in addition to programming skills and experience with higher level languages for application development. The knowledge and experience required deprive non-experts of the ability to prototype IoT applications without guidance from programming experts. The TILES Project aims at developing tools to facilitate the design, prototyping, implementation and deployment of IoT systems for non-experts without reducing application potential or diminish the application capabilities.

In early phases of the TILES Project a prototype toolkit infrastructure was implemented [28], and later extended and evaluated [11] [23]. Building on the

¹ <http://tilestoolkit.io/>

groundwork for this project, it is time to fulfill the initial goal of the TILES Project; to allow non-experts to create complex and distributed physical interfaces based on everyday object augmentation. In achieving this the TILES toolkit requires a substantial lift from an early toolkit prototype into a mature set of tools that works seamlessly together to accomplish the common goal of supporting non-experts in IoT application prototyping.

1.2 Context

This thesis is part of the TILES Project at the Norwegian University of Science and Technology. The TILES toolkit, developed for the TILES Project, aims at facilitating development of IoT applications for people with little or no programming experience. Previous work with the toolkit has successfully established a working infrastructure for device management and communication through abstraction of responsibilities into three layers: TILES Square, TILES Gateway and TILES Cloud [28]. Several studies have previously been conducted in parallel within the TILES Project [23] [30] to determine the capabilities of the TILES toolkit, and to identify future research opportunities.

The specialization project [23] has evaluated the TILES toolkit and defined strength and weaknesses with the current implementation of the toolkit. It has laid the groundwork for future research and set the course for the research conducted in this thesis project. Parallel with this research in the TILES Project, the implementation of an improved TILES Gateway smartphone app was organized [1] with the goal to add additional features to the gateway. One of the main requirements for the development of the new TILES Gateway was to extend the gateway implementation in parallel with the research explained in this paper to take advantage of the new tools introduced as a part of this research.

1.3 Problem elaboration

The problem addressed in this thesis originates from the need to bridge the gap between the TILES toolkit ideation phase and the prototyping phase. The TILES toolkit card game has been developed to enable non-experts to develop functional IoT application ideas by expressing their creativity with the tools of the TILES toolkit [19] [22]. The TILES Cards have been tested and tailored to fit the non-experts need, but the transition from the ideation phase to the prototyping phase has to be improved. At the same time, until now non-experts have required a lot of assistance from toolkit experts in prototyping, testing and deploying their IoT applications. In this report, these issues are addressed in order to transform the TILES toolkit into a self-supported system, enabling non-experts to employ the toolkit on their own, with minimal support from toolkit experts.

Additionally, when extending the TILES toolkit with new capabilities the carefully constructed facilitation mechanisms of prototyping IoT applications might not be properly considered and the support for non-experts will be diminished. Therefore, this thesis will also address the problem of customizing the TILES toolkit in such a way that the non-expert facilitation mechanisms are kept in the new extended toolkit.

1.4 Research Questions

RQ1. How to support rapid prototyping and deployment of IoT applications using TILES?

RQ1.1. How to support rapid prototyping of ideas created with TILES Cards?

RQ1.2. How to support TILES application development by non-experts?

RQ1.3. How to support toolkit extension for expert users with minimal efforts without breaking development support for non-experts?

1.5 Research Method

At the beginning of this research project, the research questions were coined together with project supervisors based on the current state of the TILES toolkit, the vision for the TILES Project and the results from the specialization project [23]. The research questions will be answered in this report by investigating the TILES toolkit through *Design Science Research* [9] and review of literature and related work. The related work chapter will be used in this thesis to assess the innovativeness of the work conducted in this research. The results and suggested future work from the specialization project [23] and results of the conducted workshops have also been considered.

The research conducted in this project had several phases. Starting from the *Knowledge Base* constructed during the specialization project, two support processes were introduced into the TILES toolkit by tapping into the *Knowledge Base* as illustrated on *Figure 1-1*. This cycle used support literature and findings from earlier research within the TILES Project to construct a structural set of process instructions to support the usage of the TILES toolkit.

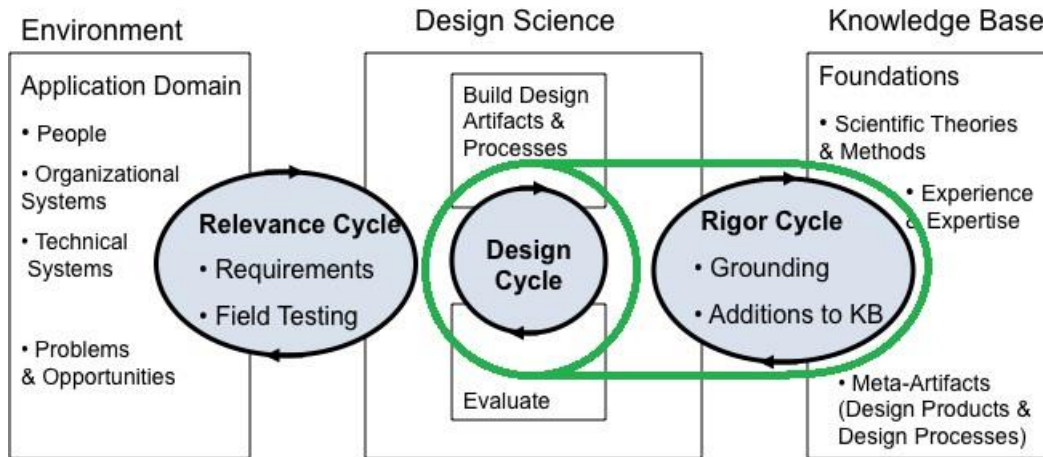


Figure 1-1, Design Science Research, Knowledge Base (adopted from [9])

The second, and most extensive phase, was the iterations of *Design Science Research* illustrated in *Figure 1-2*. Here, the extended TILES toolkit was implemented through several iterations of the *Design Cycle*. The results and conclusion of the specialization project was very important in this work as it enabled this project to tap into the *Relevance Cycle* very early in the project. Analyzing the results of the workshops from the specialization project enabled early definition of functional requirements for an extended TILES toolkit. These requirements were implemented through *Build Design Artifacts & Processes* and was iteratively tested and *Evaluated* by the TILES Gateway developers [1], which provided constant feedback on the extended TILES toolkit. In addition, two workshops and a focus group conducted at the end of this project resulted in two separate iterations tapping into the *Environment* in order to *Evaluate* the two new processes and the extended TILES toolkit.

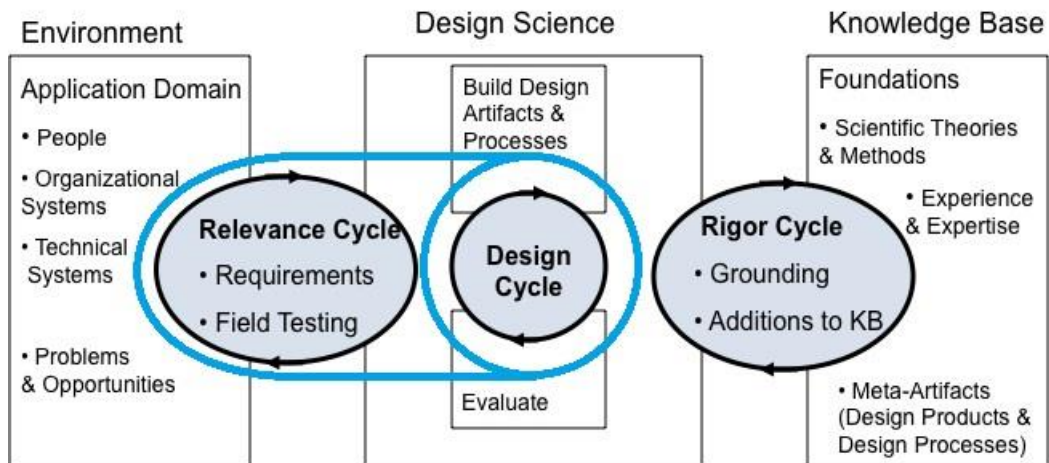


Figure 1-2, Design Science Research, Environment (adopted from [9])

The *Research Process* adapted from Oates [25] with my personal process indicated in red, can be seen in *Figure 1-3*. This figure highlights the main *strategies*, the *data generation methods* and *data analysis* methodology applied during this research project.

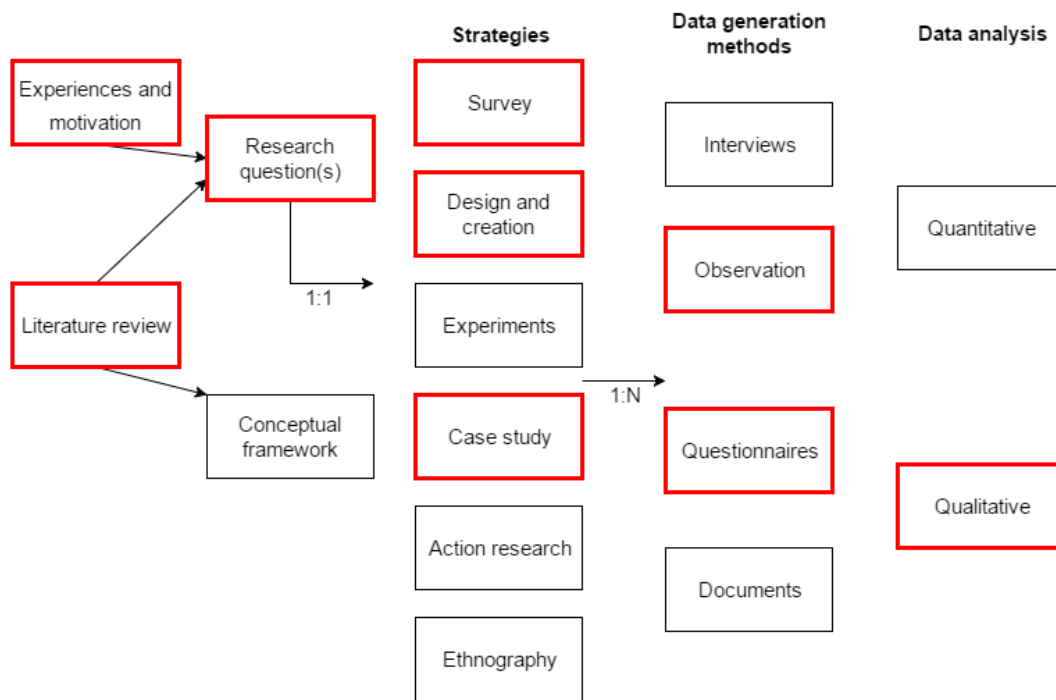


Figure 1-3, Model of The Research Process (adapted from [25])

1.6 Results

The main result of this thesis project is the *TILES toolkit Application Development Process* and the *TILES toolkit Extension Process*, which are processes developed to support both non-experts and expert users in their projects with the TILES toolkit. The processes and all their steps are detailed in the *TILES toolkit Documentation Section*, defined in this research project, which enables users to employ the processes by following the step-by-step guide hosted on the TILES Cloud web portal. The processes and the support documentation have been designed to be self-supporting to allow users to employ them without the need of toolkit experts. The documentation contains general information, process description and code samples to answer all the questions a user might have during their projects.

Introducing the processes into the TILES toolkit also required an extensive improvement of the architecture and implementation of the system. Therefore, this research project has resulted in several new components and many modifications to the existing components of the architecture. In addition, a user study conducted as several workshops and a focus group was organized to test the processes and new implementation of the TILES toolkit. Finally, this thesis serves as a contribution to the available literature on the topic of developing prototyping platforms for non-experts.

1.7 Outline

Chapter 1 has introduced the research of this thesis project. The motivation behind the research, the context of the project, the overall research questions and the method of which the research has been conducted is described in this chapter, before an overview of the results are presented.

Chapter 2 presents an overview of the development phases of the TILES toolkit in addition to introduce the tools available during the two phases. This chapter represents the current state of the TILES toolkit as it was prior to this thesis project.

Chapter 3 builds on the concepts introduced in chapter 2 by introducing two new processes: the *TILES toolkit Application Development Process* and the *TILES toolkit Extension Process*. These processes are user oriented processes, describing how to best employ the toolkit in a project. The *TILES toolkit Documentation Section* is also introduced in this chapter, which is a detailed documentation of the two new processes.

Chapter 4 introduces the *requirements* for the new extended TILES toolkit supporting the new processes. The *functional requirements* are detailed together with some *User Requirement Specifications* for guiding the graphical implementation of the extended toolkit.

Chapter 5 encapsulates the *design* of the extended TILES toolkit by presenting design elements for the architectural components of the extended system, together with requirement allocations in order to map the functional requirements to the components of the system.

Chapter 6 is dedicated to detail the technical implementation of the extended TILES toolkit. This chapter contains code snippets, explanation and requirement justification for every functional requirement of the extended TILES toolkit introduced in the previous chapter.

Chapter 7 explains the *evaluation* that was conducted in this research project. Two independent workshops and a focus groups with their results are presented.

Chapter 8 will present and describe *state-of-the-art* technology within the field of IoT development toolkits relevant for this research project. This is the *related work* section that will put the research of this project in context with other related research projects.

Chapter 9 concludes this research project by summarizing the report and presenting the results. The connection between the results and the initial research questions is an essential part of this chapter. This chapter will also debate the research conducted in this project, highlighting strength and weaknesses, and suggest topics for further research within the TILES Project.

2 TILES toolkit

The TILES toolkit is defined as “*a toolbox to support the iterative process of building prototypes of interactive objects providing design and prototyping tools*” [20]. Employing the TILES toolkit in a project is a two-phase process. The two phases are the *ideation phase* and the *prototyping phase*. This chapter is divided into two sections devoted to introduce the two phases and the tools available to the users during both phases of development. The first section, *chapter 2.1*, will introduce the *ideation phase* and the tools available in the TILES toolkit to support it. The second section, *chapter 2.2*, will similarly introduce the *prototyping phase* and the tools available to users during this phase of their projects. The tools introduced in this chapter represents the current state of the TILES toolkit. In *chapter 3*, the TILES toolkit is extended with some important new components that will bind the two phases together to ensure a smooth transition between the two phases of development.

2.1 Ideation phase

The first phase of developing an application using the TILES toolkit is the *ideation phase*. The tool available to users during this phase is the *TILES Card game* [19]. The *TILES Card game* consist of the *TILES Idea Generator Board*, seen in *Figure 2-1*, a variety of *TILES Cards*, seen in *Figure 2-2*, and a description of the ideation process that can be seen at the bottom of the *TILES Idea Generator Board*. When employed, the ideation process will systematically guide the non-expert users through several steps of conceiving an IoT application idea, by enabling the non-experts to express their creativity by combining multiple *TILES Cards* and place them on the *TILES Idea Generator Board*. The ideation phase has successfully enabled non-experts with no knowledge of IoT to learn and understand the concept in addition to develop their own fully functional IoT application idea. The TILES *ideation phase* has already been the subject of several research papers and user studies [19] [22] [23] [30]. Therefore, this thesis will not investigate any further into the *ideation phase* itself, but focus on the *prototyping phase* and the transition between the phases.

Tiles Idea Generator

Ideas

TRIGGERS
What actions trigger the object?
The object might respond to types of different types, other than human interaction, sensors, or data from remote sources.

WHAT CARDS TO USE?
Memory Action Data Feedback

THINGS
What objects are needed for your users and can help them solve the needs you have identified?
Start with a Thing card and place it in the center of the board. If you see an opportunity to use multiple objects to solve the problem, you can add additional cards, and use the Connector cards to connect them.

WHAT CARDS TO USE?
Things Connectors

RESPONSES
How does the object communicate back to the user?
The response can be direct feedback through the object itself or a user by sending data to an app or remote service.

WHAT CARDS TO USE?
Feedback Data Feedback

Storyboard

What is it like to use the object? How, where and when is it used? What does the user feel and think? Show what the object looks like, and how someone might go about using it.

Themes

Think of some new ways to frame the problem to help you think creatively. Use the themes to discuss the problem of your concept, and try to make your ideas a bigger picture.

Reflection

When you have an idea formed, use the Criteria cards to reflect on it. The criteria can reveal strengths and weaknesses of your concept and help you find ways to improve it.

Playbook

- 1** Agree, time: 5 min
Start with a user and a context that you have agreed on before.
- 2** Agree, time: 15 min
Pick through the Things cards for the cards that you want to use. Think about the cards in the center of the board. You can use multiple things if you want. How many new things can you connect with the cards you have chosen?
- 3** Agree, time: 15 min
Pick through the Triggers cards for the cards that you want to use. Think about the cards in the center of the board. You can use multiple things if you want. How many new things can you connect with the cards you have chosen?
- 4** Agree, time: 15 min
Pick through the Responses cards for the cards that you want to use. Think about the cards in the center of the board. You can use multiple things if you want. How many new things can you connect with the cards you have chosen?
- 5** Agree, time: 15 min
Pick out the ideas that you like best. Think about the cards in the center of the board. You can use multiple things if you want. How many new things can you connect with the cards you have chosen?
- 6** Agree, time: 15 min
Pick out the ideas that you like best. Think about the cards in the center of the board. You can use multiple things if you want. How many new things can you connect with the cards you have chosen?
- 7** Agree, time: 15 min
Pick out the ideas that you like best. Think about the cards in the center of the board. You can use multiple things if you want. How many new things can you connect with the cards you have chosen?

Figure 2-1, TILES Idea Generator Board
Credit: Simone Mora



Figure 2-2, TILES Cards
Credit: Simone Mora

2.2 Prototyping phase

The second phase of developing an IoT application with the TILES toolkit is the *prototyping phase*. This phase is intended to provide non-experts with the tools required to prototype their ideas generated in the *ideation phase*. The tools available in the *prototyping phase* are the *TILES Squares*, *TILES Gateway*, *TILES Cloud* and *TILES Client (client libraries)*. The specialization project [23] showed how non-expert users were able to use these tools to prototype their own simple, yet fully functional, IoT applications in IoT workshops organized by TILES toolkit experts. The following subsection will provide a technical overview of the components of the TILES toolkit. In *chapter 3* some additional tools will be introduced into the toolkit to support this phase of the development and the transition between the two phases.

2.2.1 Toolkit architecture

The architecture of the TILES toolkit is divided into three layers with clearly defined interfaces between the layers. This multilayered software architecture enables using the different layers for allocating responsibilities of the toolkit. By maintaining clearly defined interfaces between the layers, the layers can be modified, extended and even completely exchanged with little or no impact to the other layers of the architecture.

In *Figure 2-3*, we can see the layers of the TILES toolkit and how they are connected. We can see that the TILES Squares will communicate with the TILES Cloud through the TILES Gateway. Client libraries can also be connected with the cloud in order to run custom code that can interact with squares by sending and receiving events through

the layers of the infrastructure. At the bottom of the figure, we can see a *user bar*, indicating what type of users will be interacting with the various layers of the architecture. We can see that *application developers* will be interacting with the *client libraries* and *TILES Cloud* seen to the left of the figure, while the *end user* will be interacting with the *TILES Squares* and *TILES Gateway* depicted to the right of the figure.

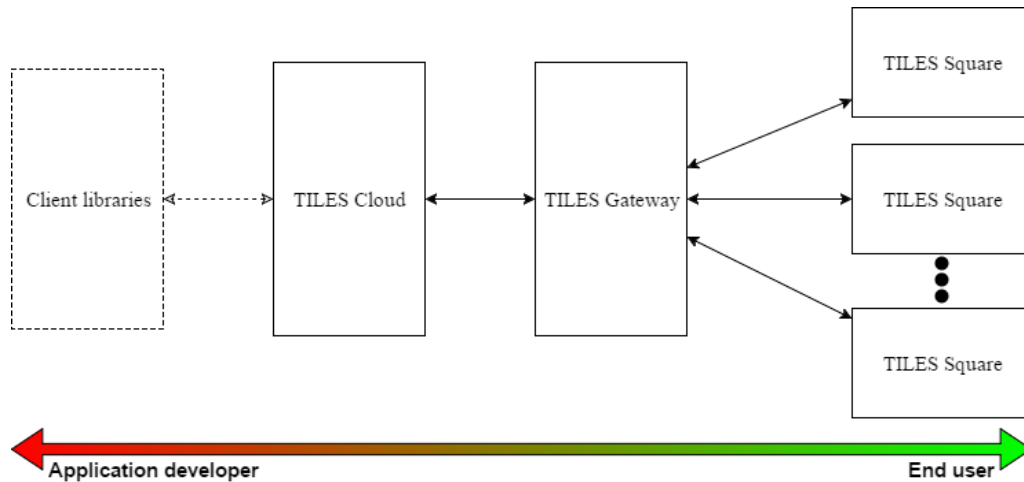


Figure 2-3, TILES architecture overview with “user bar” [23]

2.2.1.1 TILES Squares

In the heart of the TILES toolkit, we find the TILES Squares. These squares are small, embedded hardware devices that can be attached to everyday objects in order to make them *smart*. TILES Squares operate as the bridge between the real world and the digital domain of the TILES application. During the lifetime of the TILES Project, several prototypes of the TILES Squares have been developed. The first prototype was built using RFDuino² shields and can be seen in *Figure 2-4*. This prototype consists of nothing more than premanufactured RFDuino shields with a variety of hardware components that can be plugged together. These shields are excellent tools for firmware development and research of the capabilities of the TILES Squares and are still being used during research and extension of the TILES toolkit, but the shields are bulky and not very user friendly thus not suitable for TILES application prototyping.

² <http://www.rfdduino.com/>

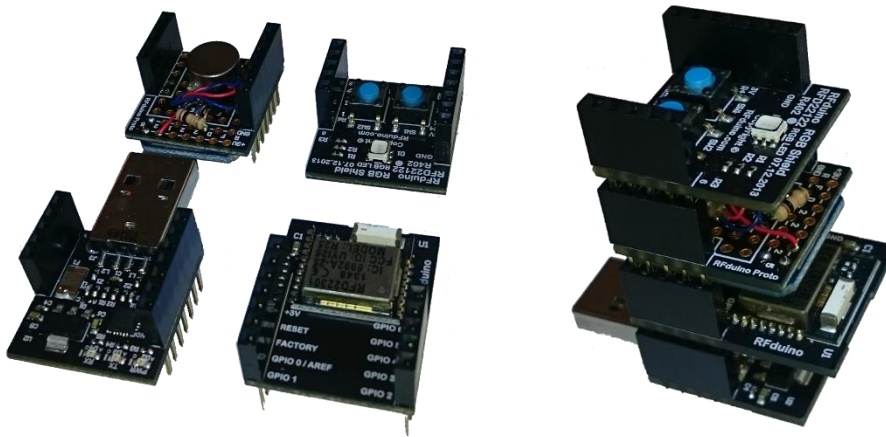


Figure 2-4, (a) RFDuino shields, (b) Connected RFDuino shields forming first TILES Square prototype

The latest prototype of the TILES Squares can be seen in Figure 2-5. This new prototype has the same hardware capabilities as the RFDuino shields above, but the hardware in the newest prototype is integrated on a single chip. This makes the square smaller and easier to attach to everyday objects, and together with a 3d-printed case the circuit hardware becomes less intrusive to the user. The hardware of the TILES Square consist of an accelerometer and a touch controller for *input primitives*, and an LED and haptic transducers for *output primitives*. The square is powered by a rechargeable battery, which can be charged with a regular micro USB cable, and it will communicate with the TILES Gateway over Bluetooth with the BLE transceiver seen on the circuit board. The latest prototype of the square also features custom extension ports and an I2C³ port for custom primitives. The I2C port enables a wide variety of third party low-speed peripherals to be plugged into the TILES Square, and together with the extension ports they support a high degree of customization for many different application scenarios, as the Squares can be customized to fit nearly any purpose.

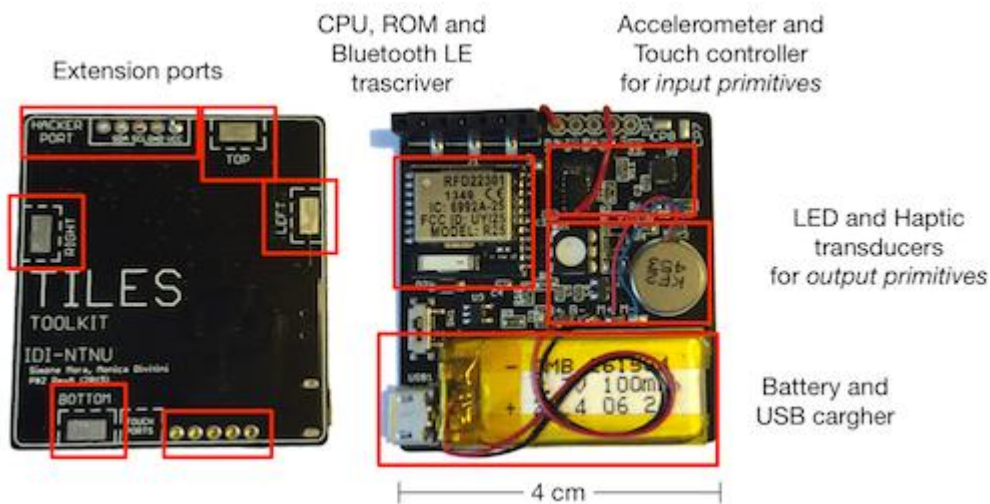


Figure 2-5, TILES Square, RFDuino, embedded hardware [20]

³ <https://www.i2c-bus.org/i2c-bus/>

The TILES Squares are designed such that no application logic is running on the Squares themselves, but will simply forward all detected events to the connected TILES Gateway. This abstraction feature enables non-experts to prototype applications without having to deal with firmware development of the TILES Squares using the Arduino⁴ IDE. *Table 2-1* shows the interaction primitives implemented in the latest firmware of the TILES Squares. These primitives are available to non-expert developers in the *client libraries* without the need to program the expected behavior in the firmware, which is an important facilitation measure of the TILES toolkit.

Input primitives		
Primitive	Degrees of freedom	Example mapping
Touch/Tap	Single, double	Send a command, log a quantity
Tilt	Tilted, not tilted	Select a function, binary switch
Output primitives		
Primitive	Degrees of freedom	Example mapping
LED Light feedback	on, blink, fade, off <i>Supported colors:</i> <i>>16 000 000</i>	Continuous notification about the status of process
Haptic feedback	Vibration pattern <i>Supported patterns:</i> <i>long/burst</i>	Discrete notification about the status of process

Table 2-1, TILES Square interaction primitives [23]

2.2.1.2 TILES Gateway

Since the TILES Squares are not able to connect directly to the Internet, the gateway layer is needed between the squares and the cloud infrastructure. The TILES Gateway layer is developed using Ionic⁵, Cordova⁶, a cross-platform development tool for smartphones. Cordova enables smartphone application development using JavaScript, CSS and HTML, without the need to be familiar with native programming languages for iOS⁷ or Android⁸. The TILES Gateway features a user interface for connecting to the TILES Cloud infrastructure, and for discovering and connecting to the TILES Squares. After connecting to a square, the gateway will be responsible for registering the square to the cloud server, in addition to subscribe to commands targeting this specific square.

⁴ <https://www.arduino.cc/>

⁵ <https://ionicframework.com/>

⁶ <https://cordova.apache.org/>

⁷ <https://www.apple.com/ios>

⁸ <https://www.android.com/>

In the gateway layer, communication with the TILES Squares happens over Bluetooth using the BLE protocol, while communication with the Cloud infrastructure happens over a standards IP-based internet connection using the MQTT protocol. The gateway will translate each event between the two protocols, BLE and MQTT, in both directions. The TILES infrastructure is very flexible with respect to internet connection, allowing the gateway to communicate through Wi-Fi, mobile telecommunication, or possibly any other type of internet connection. The only requirement is that the gateway is able to maintain a steady internet connection in order to communicate over the MQTT protocol without dropping the connection. An important limitation of this configuration is the limit on the size of messages supported by the BLE protocol. This protocol is limited to send only 20 bytes of data [6] per message, which means that the event and command messages between the Gateway and Squares must be short and concise.

2.2.1.3 TILES Cloud

The TILES Cloud is a publicly accessible internet server, currently available at IP *178.62.99.218*. The server is hosted using *DigitalOcean*, see *Appendix A*. The server software is written in JavaScript using Node.js⁹, with an HTTP interface at port 3000¹⁰. The TILES Cloud has an MQTT broker, allowing clients to connect for transmitting real-time events. MQTT is a publish-subscribe based, lightweight messaging protocol that runs on top of TCP/IP. The MQTT implementation of the server enables clients to subscribe and publish to *topics*, and the server will be responsible for delivering the messages to the subscribing clients. The topics are composed from the unique id of the user and the unique id of the TILES Squares. This enables clients to publish and subscribe to events to and from specific TILES Squares without knowing where they are or how the event will be delivered over the Internet. A gateway receiving an input primitive event from a TILES Square will publish a message to the MQTT broker together with the respective topic, and the broker will be responsible to distribute the message to all subscribing clients.

The HTTP interface of the TILES Cloud server is used for administration purposes. By navigating to the server URL, the user will be able to browse through the registered users and the TILES Squares registered to them. Users are also able to register webhooks to their squares, which will initiate HTTP POST request on incoming events from the configured TILES Squares.

2.2.1.4 TILES Client

Even though all events are processed by the MQTT broker of the cloud infrastructure, the cloud offers no means of application development other than support for registering

⁹ <https://nodejs.org/en/>

¹⁰ <http://178.62.99.218:3000/>

webhooks to be triggered at incoming events. This means that the TILES toolkit needs a *fourth layer* for implementing application logic, hence the TILES Client.

The TILES Client is a JavaScript SDK for TILES application development. The SDK enables developers to write application logic in pure JavaScript, and both run and debug the code using Node.js. The SDK contains a number of JavaScript APIs for facilitating the development process of TILES applications. Code templates explaining the APIs are also provided in the SDK, allowing non-experts to quickly get started with prototyping their TILES applications. The JS APIs are subject of the user study explained in the specialization project [23].

Like the TILES Gateway, the TILES Clients will connect to the MQTT broker of the TILES Cloud. The JS APIs provided in the TILES Client JavaScript SDK provide code templates for connecting to the broker with only a few lines of code. The TILES Client will be subscribing to events of the TILES Squares, and will be able to process and react to the events.

3 From Ideation to Prototyping

In the previous chapter, the two phases of development with the TILES toolkit were introduced and the tools available during the phases were explained. A problem with the current structure of the TILES toolkit is that there is a gap in the available tools in the transition from the ideas generated in the *ideation phase* and the applications developed during the *prototyping phase*. The *ideation phase*, as explained in *chapter 2.1*, has defined a systematic process for guiding the users through the steps of generating an idea. Besides the actual tools for application prototyping, the TILES toolkit must offer a prototyping process, analogous to that of the idea generation, in order to drive the development forward and allow non-experts to effortlessly take part in the *prototyping phase*. Currently, employing the TILES toolkit in application prototyping requires toolkit experts to provide documentation and guidance to non-experts during the whole phase.

In this chapter, some additional tools will be introduced into the TILES toolkit, which will enable non-experts to effortlessly take part in the whole development, both in ideation and prototyping, of IoT applications. Construction of the new tools elaborated in this chapter, was performed iteratively through an initial set of iterations of the *Design Cycle* seen in *Figure 1-1, Design Science Research, Knowledge Base (adopted from [9])*, where results of the specialization project [23] enabled early iterations of tapping into the *Knowledge Base* through the *Rigor Cycle*.

3.1 TILES toolkit Processes

In order to support the transition from ideation to prototyping in the TILES toolkit, the *TILES toolkit Application Development Process (TADP)* was introduced into the toolkit. A major concern of the TILES Project has always been on supporting the non-expert developer in transforming ideas into IoT application prototypes, and the TADP is constructed with this exact purpose. However, it is important to point out that support for the expert user is of equal importance in order to let the expert user create customized toolkits for specific usage areas, without breaking the carefully constructed support and facilitation mechanisms for non-expert prototyping. For this reason the *TILES toolkit Extension Process (TEP)* was introduced into the TILES toolkit, intended to enable expert users to tailor the toolkit to their specific needs, yet keep the

support for the non-experts in prototyping performed with the extended toolkit. The following two subchapters are devoted to introducing the TADP (*chapter 3.1.1*) and TEP (*chapter 3.1.2*). In addition, in order to support and promote their usage and provide a stepwise description of the processes, the *TILES toolkit Documentation Section* (TDS) was constructed, which is elaborated in *chapter 3.2*.

3.1.1 TILES toolkit Application Development Process

In their paper, Mora et al. have defined a five-step application development process using RapIoT [21]. This process has been refined in this thesis into the *TILES toolkit Application Development Process* (TADP), which is designed to fit together with the TILES Card ideation process to allow non-experts to transform their ideas into working TILES application prototypes using the TILES toolkit. By adapting the RapIoT development process and refine it to fit the TILES toolkit, the TILES toolkit will inherit the qualities, such as *application appropriation*, from the RapIoT toolkit. The refined TADP and its eight steps can be seen on the *next page*, while *Figure 3-1* shows an illustration of the whole process.

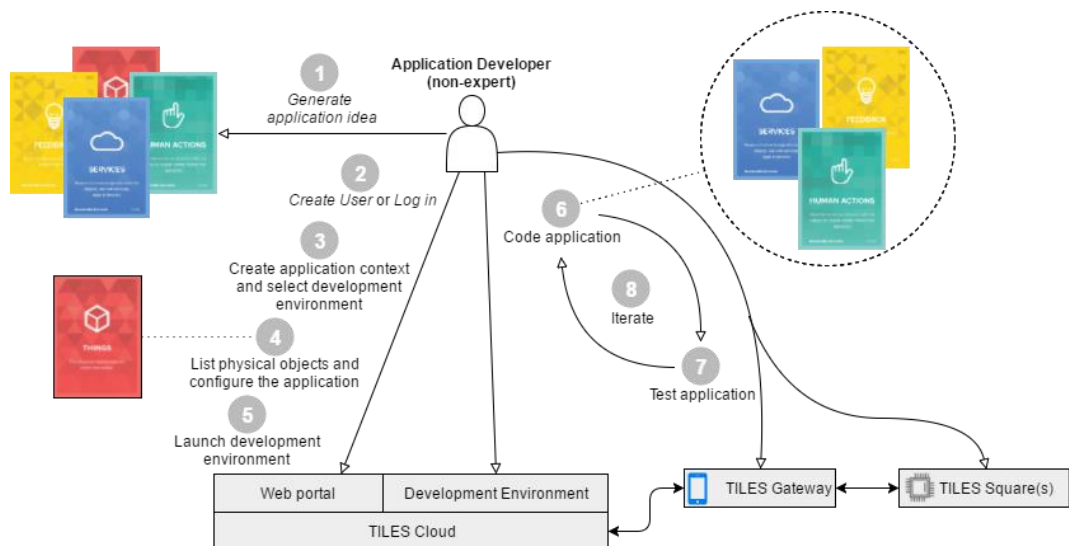


Figure 3-1, TILES toolkit Application Development Process

1. **Ideation Phase** – involves employing the TILES toolkit ideation process with the TILES Card game to develop an IoT application idea.
2. **Create User** – entails creating a user in the TILES Cloud web portal.
3. **Create Application** – involves defining an application name and create the application context and selecting development environment in the TILES Cloud web portal.
4. **List Physical Objects** – entails using the TILES Cards to identify the physical objects needed in the application, and listing them by configuring the application from step 3 with abstract placeholders for your physical things.
5. **Launch Development Environment** – entails starting up the selected development environment and navigating to the starting point of the environment.
6. **Code Application** – involves coding the program behavior by mapping the TILES cards into executable program code with the steps listed below.
 - a. Map HUMAN ACTIONS cards with TILES API events
 - b. Map FEEDBACK cards with TILES API commands
 - c. Map SERVICES cards with TILES API sources
 - d. Use TILES API for additional application behavior
7. **Test Application** – involves starting application and using gateway to discover and use TILES Squares in your application by following the steps below.
 - a. Run application in test mode
 - b. Procure and ready TILES Squares
 - c. Procure and ready Physical Objects
 - d. Open TILES Gateway app and log in
 - e. Select the application in TILES Gateway
 - f. Pair TILES Squares with abstract placeholders
 - g. Use the application
8. **Iterate step 6-7** – entails looping through step 6 and step 7 until desired application behavior is accomplished.

3.1.1.1 Supporting TILES Cards

As seen above, step 4 and 6 of the TADP are constructed based on the usage of the TILES Cards, and will be responsible to bridge the gap between the ideation phase and the prototyping phase. These steps entails using the *THINGS* cards for identifying physical objects, and *HUMAN ACTIONS*, *FEEDBACK* and *SERVICES* cards to identify the capabilities of the objects in the application. The TDS contains detailed explanation and examples on how to transition from the cards to executable program code. It is important to note that the TADP does not impose using the TILES Cards for idea generation, but special emphasis is put on support for the cards, thus using the cards is a highly recommended tool in the ideation phase.

In step 4 of the TADP, the users will be guided through the steps of mapping physical objects to *abstract objects*, or *digital placeholders* for physical objects, by configuring them in their applications. This is the first step towards transitioning from ideation and

into the tangible user interface that is a TILES application. The transition from ideation to tangibility undertakes three steps that can be seen in *Figure 3-2*. This figure illustrates how the representation of objects changes from the ideas on the *TILES idea generator board*, through a digital representation of the objects, before emerging as tangible physical objects in the IoT application prototype.

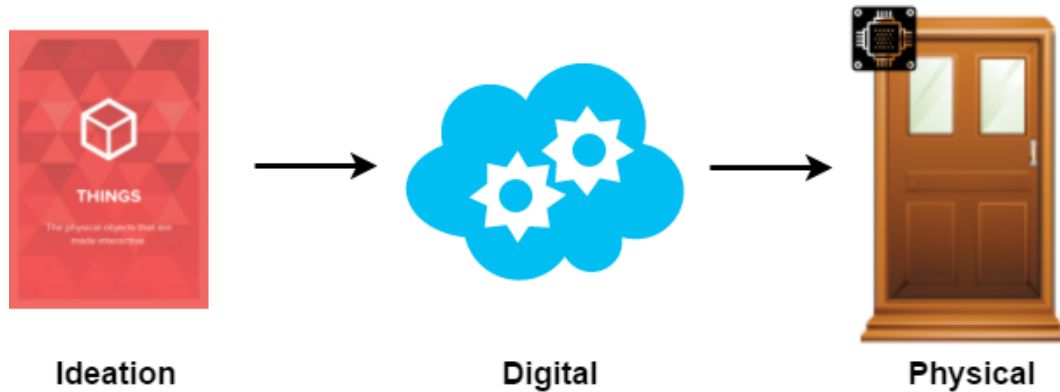


Figure 3-2, TADP object representation flow

A similar transformation occurs with the application capabilities as *HUMAN ACTIONS*, *FEEDBACK* and *SERVICES* cards are transformed from the ideation phase, through a digital representation in the program code, before the user can interact with the things in a tangible user interface.

3.1.1.2 Users

The TADP is intended to be used by *non-experts*, to enable people with little or no programming experience to successfully and quickly prototype their IoT applications based on everyday object augmentation. The process can either be employed in a fixed workshop environment, like the once described in the specialization project [23], or be utilized by non-experts themselves in their DIY projects with no supervision from toolkit experts. This gives the process a dual purpose objective, which can either be a learning objective in a fixed workshop environment, or to quickly prototype IoT application ideas for production and self-entertainment. The process together with the supporting documentation in TDS is intended to be self-explanatory, and should guide the users through the whole development process after an idea has been generated until an application prototype exist.

When the TADP has fulfilled its purpose and the application prototype exist, the application is ready to be used by end users. The steps of using an application prototype that has been successfully implemented are very similar to step 7 of the TADP, but with some small modifications. The steps of using an application prototype are listed on the *next page*.

1. Open TILES Gateway app and log in with the owner of the application
2. Select the application you want to run and see the list of physical objects needed
3. Procure the TILES Squares needed as seen in the list from step 2
4. Procure the physical objects needed as seen in the list from step 2
5. Pair the TILES Squares with the placeholders in the list from step 2, and attach the Squares to the physical objects to be used
6. Start the application from the Gateway app
7. Use the application

3.1.2 TILES toolkit Extension Process

Since the TILES Squares and the TILES development APIs only support a limited number of input and output primitives, and the TILES Project aims at being applicable to a broad specter of IoT scenarios, extension of the toolkit must be supported. The layered structure of the TILES toolkit already supports any type of event to propagate the TILES infrastructure without the need to make changes to the core layers. However, in order to add additional hardware capabilities to the TILES Squares and support new interaction primitives in the JS SDK, the implementation of the toolkit must be altered. Extending the toolkit is a complex task as it requires knowledge of firmware programming using the Arduino IDE to develop and flash program code to the TILES Squares. We cannot expect non-experts to be familiar with such complicated development procedures, thus yet another process is needed, intended to guide the experts in the extension of their own custom TILES toolkit. The *TILES toolkit Extension Process (TEP)*, which support extension of the TILES toolkit, is listed below with all its steps and is illustrated in *Figure 3-3*.

1. *Device Development* – involves extending the hardware prototype, possibly with additional hardware, and implement input/output primitives in firmware.
2. *Library Development* – involves implementing new features to the toolkit APIs in order for application developers to be able to utilize the capabilities created in step 1.
3. *API Deployment* – entails deploying the extended libraries to the TILES Cloud server.
4. *TILES Card deck creation (optional)* – involves creating a deck of TILES Cards by removing all cards that are not supported by the extended hardware configuration of the TILES Squares and possibly creating additional cards for the new interaction primitives.

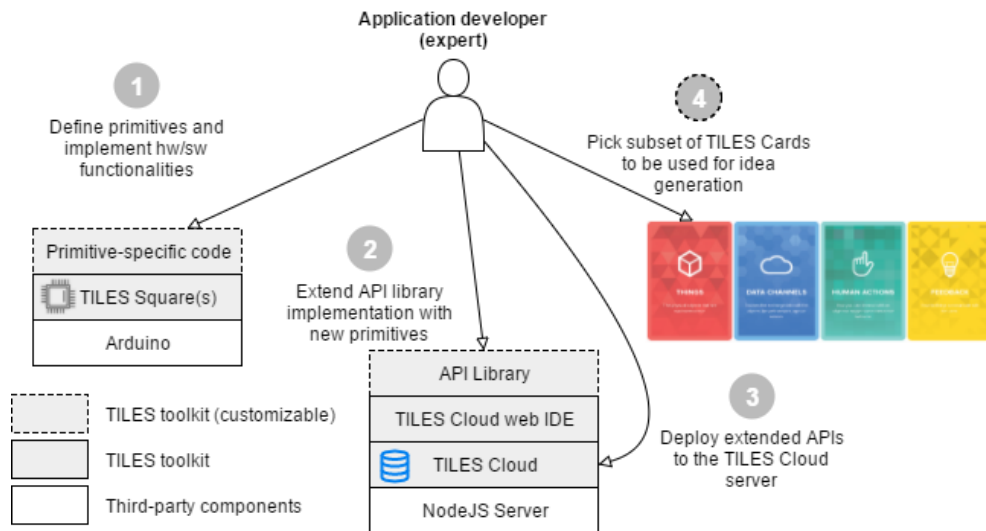


Figure 3-3, TILES toolkit Extension Process

3.1.2.1 Users

The intended users of the TEP are computer science students with some programming knowledge and experience with development, and are referred to as the *expert users* or simply *experts*. The experts should have experience with hardware development or C-programming in addition to experience with higher level scripting languages such as JavaScript. Whenever the need for additional hardware capabilities of the TILES Squares or new functionality in the APIs arise, the expert user should employ the TEP to extend the TILES toolkit. The last step of the TEP is an optional step employed to construct a subset of the TILES Cards to be used for ideation with the new extended toolkit. This step ensures that only the supported interaction primitives and services are available in the TILES Card deck, and it will encourage the use of the TILES Cards in ideation for non-experts. By removing the cards that are not supported by the extended toolkit, the experts will ensure that TILES Cards and TADP can be employed by non-experts in ideation and prototyping with minimal risk of inconsistency due to unsupported hardware capabilities. The detailed instruction on how to employ the TEP is explained in the TDS that will be introduced in the following subchapter.

3.2 Toolkit Documentation Section

The two TILES toolkit processes, introduced in this chapter, identifies the users of the toolkit, the steps of developing an application in TADP, and the steps of extending the toolkit in TEP. In order to promote the usage of the processes and provide a step-by-step guide for employing them, the *TILES toolkit Documentation Section* (TDS) has been constructed. The TDS will be hosted in the TILES Cloud web portal, which ensures that the TDS has the same availability as the TILES Cloud server, and makes the TDS available during all phases of interacting with the TILES toolkit. The hosted

TDS can be seen on the *Docs* section on the web portal¹¹, and is also available in *Appendix*. The TDS is intended to be used for understanding how to apply the TADP and TEP, and the following subchapter will introduce the outline of the TDS.

3.2.1 Outline

The TDS has been divided into *five* chapters as seen in *Table 3-1* of the outline of the TDS. The TDS consist of a step-by-step guide explaining the usage of the TILES toolkit. Each chapter of the TDS is briefly introduced in the flowing subchapters.

3.2.1.1 Getting Started

The Getting Started chapter of the TDS will introduce the documentation, explain the outline, and elaborate what section is relevant for the user depending on their objective with the TILES toolkit. This chapter will also introduce the main components of the TILES toolkit and provide some important information about the available tools. *Getting Started* is intended for non-experts and experts alike, and will most importantly give instructions on how to proceed depending on the user's intention with the toolkit. Non-experts are encouraged to study all sections of this chapter, while expert users should only need to get familiar with its content and revisit the chapter later if needed.

3.2.1.2 Application Development Process

This chapter of the TDS is devoted to guide the user through the TADP with a separate section dedicated to each step of the process. By passing through this section sequentially, the user should be able to follow a carefully constructed step-by-step explanation for transitioning from the ideation phase to prototyping the application. This chapter is not only concerned with the actual program implementation, but all required steps of the process of rapidly prototyping an IoT application.

3.2.1.3 JavaScript API

This chapter of the TDS is a detailed explanation of the TILES toolkit JavaScript APIs, with samples on how to transform the TILES Cards into executable program code for a TILES application. This chapter is built with the non-expert in mind, but expert users should also be able to find this chapter useful when prototyping their TILES applications.

¹¹ <http://178.62.99.218:3000/docs>

3.2.1.4 TILES toolkit Extension Process

This chapter of the TDS is dedicated to the more advanced process of the TILES toolkit. The intended audience of this chapter is the expert users that wants to employ the TEP in extending the TILES toolkit. This chapter will contain a more advanced language as the target users are expected to be more experienced.

3.2.1.5 Rule Engine API

Similar to the JavaScript API chapter, this chapter of the TDS will explain the Rule Engine API. It will put special emphasis on the usage of the TILES Cards for transitioning from ideation to prototype, but both non-experts and expert users should find this chapter helpful while prototyping applications using the Rule Engine API.

- **Getting Started**
 1. Introduction
 2. Getting started
 3. TILES toolkit
 4. Development environments
 5. TILES Square primitives
- **Application Development Process**
 0. Process description
 1. Ideation Process
 2. Create User
 3. Create Application
 4. List Physical Objects
 5. Launch Development Environment
 6. Code Application
 7. Test Application
 8. Iterate step 5-6
- **JavaScript API**
 1. Introduction
 2. Event Reader API
 3. Map HUMAN ACTIONS
 4. Map FEEDBACK
 5. Map SERVICES
 6. Example scenario 1
 7. Example scenario 2
- **TILES toolkit Extension Process**
 0. Process Description
 1. Device Development
 2. Library Development
 3. API Deployment
 4. TILES Card Selection
- **Rule Engine API**
 1. Introduction
 2. Map HUMAN ACTIONS
 3. Map FEEDBACK
 4. Map Services
 5. Example scenario

Table 3-1, TILES toolkit Documentation Section outline

4 Requirement Specification

In the previous chapter, some new tools were defined intended to support the usage of the TILES toolkit in different contexts. In order to integrate these tools into the TILES toolkit, the implementation of the toolkit has to be modified. This chapter will specify and elaborate the *requirements* for the extended TILES toolkit. The corresponding *design* and *implementation* of the requirements are detailed in the following two chapters. The requirement specification explained in this chapter is part of the *Rigor Cycle* tapping into the *Knowledge Base* as illustrated in *Figure 1-1, Design Science Research, Knowledge Base (adopted from [9])*. The requirements are derived from the research questions and the vision for the TILES Project, together with the well tested and robust *system requirements*, seen in *Table 4-1*, established in the specialization project [23]. Adding support for the *system requirements* is important to ensure the continuous support of the workshop template designed in the specialization project. Reusing the results from the specialization project enabled very early iterations of the *Rigor Cycle* during this project, which was imperative for the success of the extensive development performed in this project.

ID	Description	Priority
SR1	The toolkit should support interaction with TILES Squares through either TILES Gateway, or a platform specific gateway provided by the toolkit.	H
SR2	The toolkit should support running custom application code in JavaScript using Node.js in such a way that existing TILES Client code can be run with minimal effort.	H
SR3	The toolkit should provide an administration web interface for maintaining users and connected devices, and for relaying events between the other components of the infrastructure.	H
SR3.1	The toolkit web interface should provide the ability to create simple IoT applications in the cloud, without running client code.	M
SR3.2	The toolkit web interface should provide extension points to other web based services, such as IFTTT.	M
SR4	The toolkit should feature multiple gateways, running on multiple platforms (smartphone, RPI, etc.).	M
SR5	The toolkit should support integration with custom embedded non-ip hardware devices (non-ip devices similar to the TILES Squares).	L
SR6	The toolkit should support integration with custom embedded ip devices.	L
SR7	The toolkit should implement modern security mechanisms for providing security to users running IoT applications.	L

Table 4-1, System requirements [23]

4.1 Application appropriation

From the very beginning, the advancement of the TILES Project has been motivated by the goal to reduce the complexity of developing IoT applications, especially for non-experts. Mora et al. explain in their paper that “*Prototyping IoT systems is challenging because it requires dealing with a heterogeneous mix of hardware and software components arranged in a multi-layer architecture*” [21]. The layered architecture of the TILES toolkit, introduced in *chapter 2*, has successfully abstracted hardware and software complexity, and user studies have proven that the current level of abstraction successfully enables non-experts, with some help from toolkit experts, to develop IoT applications using the TILES toolkit [23]. The TADP, explained in *chapter 3.1.1*, is intended to bridge the remaining gap between ideation and prototyping by guiding the non-experts through the steps of prototyping their own IoT applications with the TILES toolkit. In order to support this process, however, application appropriation is essential, and some fundamental changes to the architectural layers are needed.

The requirements for application appropriation are derived from the TADP, the RapIoT process [21], and the current state of the TILES toolkit. The high priority

requirements represents the minimum set of requirements for supporting application appropriation. The requirements can be seen in *Table 4-2*. Priorities of the requirements are listed as High (H), Medium (M) or Low (L) in the column labeled ‘Pri.’. Interdependencies of the requirements are listed in the ‘Idep.’ column, while the ‘SR’ column lists the related *System Requirements* seen in *Table 4-1*.

ID	Description	Pri.	Idep.	SR
AR1	The TILES Cloud platform should support creating, listing, configuring and deleting applications (CRUD operations on applications).	H	-	SR3 SR3.1
AR2	The CRUD operations should be available through REST services and the TILES Cloud web portal.	H	AR1	SR3
AR3	The TILES Cloud platform should support abstract devices or placeholders to be created and removed from TILES applications.	H	AR1	SR3 SR3.1
AR4	Configuring abstract devices should be available through the TILES Cloud web portal.	H	AR3	SR3.1
AR5	The TILES Cloud platform should support pairing abstract devices with physical TILES Squares.	H	AR3	-
AR6	Pairing abstract and physical TILES Squares should be available through REST services at application runtime.	M	AR5	-
AR7	The TILES Gateway should be responsible for pairing the discovered physical TILES Squares with the abstract devices.	M	AR6	-
AR8	The last events and availability state of physical devices paired with an abstract device should be accessible in the TILES Cloud web portal.	M	AR5 AR6	-
AR9	The TILES Cloud platform should support publishing and subscribing to TILES Squares using the paired abstract squares of an application.	H	AR1 AR5	SR1 SR3
AR10	The TILES Cloud platform should support using the abstract devices in application development without concern for the physical hardware devices.	H	AR5 AR9	SR3.1
AR11	The TILES Gateway should support user actions such as selecting and running applications.	M	AR5 AR6	SR1
AR12	The TILES Cloud platform should support pairing abstract devices with third party embedded hardware devices.	L	AR3	SR5 SR6
AR13	The TILES Cloud platform should implement modern security mechanisms to protect the users’ applications from unauthorized access	L	-	SR7

Table 4-2, Application appropriation requirements.

4.2 Rule Engine Development Environment

The requirements for the Rule Engine development environment, seen in *Table 4-3*, are constructed based on the TADP, the research questions, the results from the evaluation of the TILES toolkit in the specialization project [23], and previous work with the TILES Project [11]. Until now, the TILES Project has focused on application development using a textual programming approach with JavaScript, but as discovered in the specialization project [23], many of the non-experts participating in the study were having problems with the fundamental concepts of textual programming languages. In order to offer broader support for development by non-experts and support rapid prototyping of ideas, the Rule Engine development environment is introduced as a tool to prototype applications without having to write textual program code.

Priorities of the requirement for the Rule Engine development environment are listed as High (H), Medium (M) or Low (L) in the ‘Pri.’ column. The ‘Idep.’ column show the interdependencies while the ‘SR’ column lists the related *System Requirements* seen in *Table 4-1*.

ID	Description	Pri.	Idep.	SR
RR1	The TILES Cloud platform should support application development by defining rules without having to write code textually.	H	-	SR3, SR3.1
RR2	The TILES Cloud platform should provide a Rule Engine API that enables expert-users to maintain a list of available TILES rules to be used by non-experts.	H	-	SR3.1
RR3	The TILES Cloud platform should support creating, configuring and deleting rules through REST services and the TILES Cloud web portal.	H	RR1	SR3
RR4	The TILES Cloud platform must keep the support for existing development environments upon introduction of the new rule based environment.	H	RR1	SR2
RR5	The TILES Cloud platform should offer a rule based approach for integrating the application with IFTTT without having to write code textually.	M	RR3	SR3.2
RR6	The Rule Engine should support the ability to automatically generate JavaScript code using the JS APIs for every Rule Engine application.	L	-	SR2

Table 4-3, Rule Engine Requirements.

4.3 TILES toolkit IDE

The requirements for creating the *TILES toolkit IDE* (TIDE) are constructed from the research questions, the vision for the TILES Project and the TADP, and they can be seen in *Table 4-4*. The TILES toolkit should support rapid prototyping by non-experts, which means downloading and setting up the local development environment with all APIs manually for each project is not a satisfactory approach. Setting up the local development environment is not a part of the workshop structure [23], as it was considered out of the scope of the non-experts expertise. By introducing the TIDE to the TILES toolkit, non-experts will be able to get started with development without requiring assistance from toolkit experts to set up the development environment. Introducing the TIDE would also move the *TILES Client* layer, elaborated in *chapter 2.2.1.4*, into the *TILES Cloud*, and the client layer would become obsolete from a non-expert's perspective.

Priorities of the requirement for TIDE are listed as High (H), Medium (M) or Low (L) in the 'Pri.' column. The high-priority requirements forms the minimum set of requirements needed to support TIDE in developing TILES application for non-experts. The 'Idep.' column show the interdependencies while the 'SR' column lists the related *System Requirements* seen in *Table 4-1*.

ID	Description	Pri.	Idep.	SR
IR1	The TIDE should run in a web browser, and be hosted by the TILES Cloud.	H	-	SR2
IR2	The TIDE should provide a textual editor for writing JavaScript code.	H	IR1	SR2
IR3	The TILES Cloud platform should support creating TIDE workspaces.	H	-	-
IR4	The TILES Cloud platform should automatically configure the workspace with the JavaScript API for all TIDE applications.	H	IR3	SR2
IR5	The TILES Cloud platform should support deployment of TIDE applications by running the JavaScript client code directly from the workspace location in the Cloud.	H	-	-
IR6	The TIDE should support debugging applications and logging output to a console window in the browser.	M	IR1	-
IR7	The TILES Cloud platform should support automatic JS code template generation for the abstract devices configured in the application context.	M	IR2	SR2
IR8	The TILES Cloud platform should support security measures to protect the private TIDE application workspaces from unauthorized access.	L	-	SR7

Table 4-4, TILES toolkit IDE requirements.

4.4 TILES Cloud web portal

Many of the functional requirements introduced in this chapter explicitly mention the TILES Cloud web portal. This entails that the UI of the TILES Cloud will need to change to accommodate the new requirements. Additionally, the TDS, introduced in *chapter 3.2*, will be hosted on the TILES Cloud web server, in order to be available for users of the TILES toolkit during their projects. The current version of the TILES Cloud web portal is a very simple website, as seen in *Figure 4-1*, that has been designed with no regards for user experience (UX). Designing a comprehensive UI considering both non-experts and expert users is out of the scope of this thesis project. However, in order to properly accommodate the TADP, TEP and TDS, and support rapid prototyping of IoT applications, some essential modifications to the UI of the web portal is necessary.

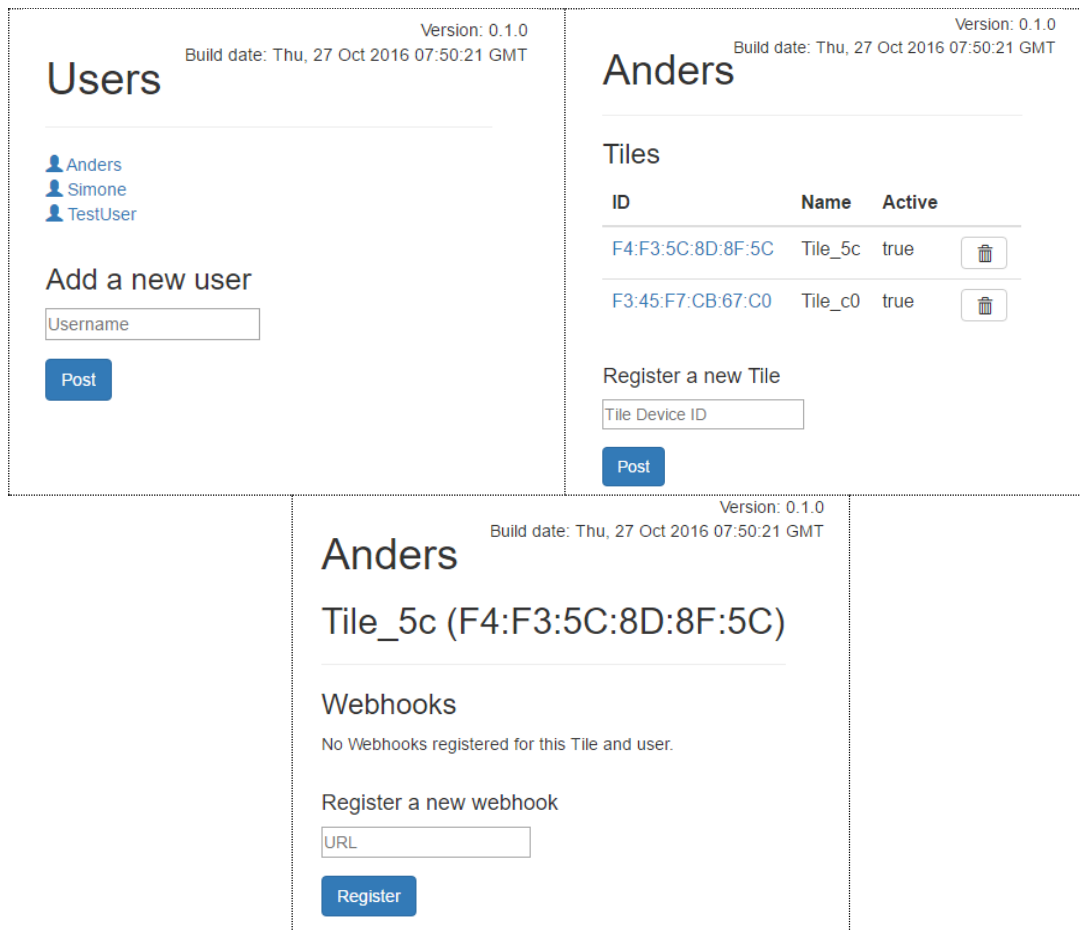


Figure 4-1, Old TILES Cloud web portal UI (a) All users, (b) Specific user with TILES Squares, (c) Specific TILES Square

The User Requirement Specification (URS) for the extended TILES Cloud web portal can be seen in *Table 4-5*. These requirements are intended only as guidelines for the implementation of the design of the web portal in order to accommodate the functional requirements listed in the previous subchapters.

ID	URS
UR1	Users should be able to navigate the website by using a navigation panel always visible at the top of the site
UR2	Users should be able to easily identify what section of the website they are seeing
UR3	The TILES Cloud web portal should host the TDS
UR4	The TDS available on the TILES Cloud web portal should maintain a separate navigation panel, enabling users to navigate the TDS separately from the rest of the website
UR5	Users should be able to go through the TDS step by step in the TILES Cloud web portal
UR6	The TILES Cloud web portal should be styled using bootstrap to ensure a familiar user experience
UR7	TIDE should be hosted by the TILES Cloud, but should be separated from the administration panel of the web portal

Table 4-5, URS of TILES Cloud web portal

5 Design

The functional requirements for the extended TILES toolkit were introduced in *chapter 4*. This chapter will be dedicated to construct the design elements of the extended TILES toolkit. First, the actors of the system will be introduced before the architecture of the TILES toolkit is extended with new components to feature the requirements from *chapter 4*, and the TADP, TEP and TDS tools from *chapter 3*. At last, the functional requirements from the previous chapter will be mapped to the architectural components of the system in order to determine the impact of the required implementation for the functional requirements, and to see what source files must be modified.

5.1 Users

The extended TILES toolkit, developed during this project, identifies two main human actors. These actors are *non-experts* and *expert users*. A use case diagram of the system with these two actors, in addition to include the *End User* actor, can be seen in *Figure 5-1*.

Non-experts, as elaborated in *chapter 3.1.1*, are users with little or no programming experience that wants to prototype IoT applications. These *non-experts* are analogous to the target users of the TILES toolkit ideation process using the TILES Cards [22]. *Non-experts* will be interacting with the TILES toolkit by following the TADP. The TDS with its explanation of the toolkit and the steps of the TADP will guide the *non-experts* as they progress in the application prototyping phase. For programming their applications, the *non-experts* will be using the TIDE or the Rule Engine Development Environment.

Expert users, as introduced in *chapter 3.1.2*, refers to people with some programming knowledge and experience with development. *Expert users*, or simply *experts*, should have some experience with hardware development or C-programming, in addition to experience with higher level scripting languages such as JavaScript. *Experts* will be interacting with the TILES toolkit by following the TEP, and using the supported documentation in TDS. The *experts* are also encouraged to use TIDE in their application prototyping, while they will most likely find the Rule Engine Development

Environment insufficient as it limits the capabilities of the TILES application. *Experts* are also the alleged user of the local development environment. Using the local development environment will present the full JavaScript API to be downloaded. Using the JavaScript API locally enables TILES Square devices to be integrated in any application using Node.js.

A third human actor, *End User*, is also identified, which can be seen in *Figure 5-1*. This actor is included in the Use Case Diagram to illustrate that the TEP and TADP supports prototyping of applications that are ready to be deployed and used by non-experts. The steps of using a TILES application was introduced in *chapter 3.1.1.2*. No predefined requirements exist for the *End User* as the TILES Squares forms non-intrusive tangible interfaces that can be used by anyone regardless of level of experience with other technical systems.

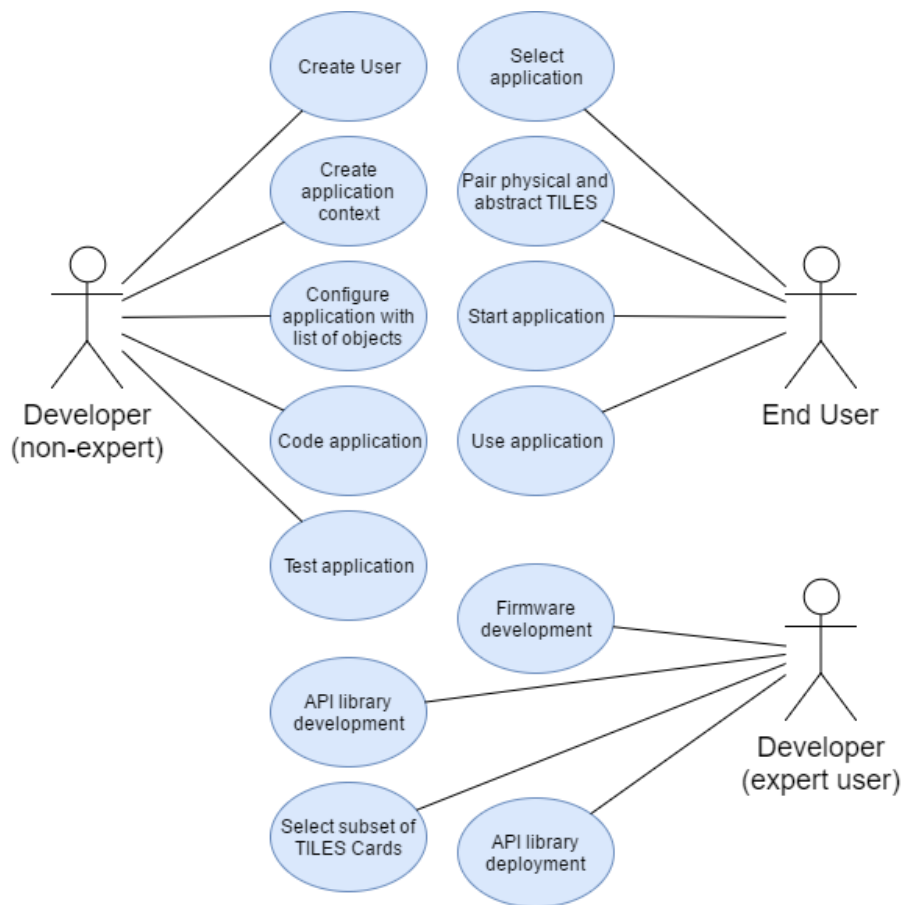


Figure 5-1, TILES toolkit Use Case Diagram

5.2 TILES toolkit revised

In *Figure 2-3*, we saw a simplified illustration of the existing architectural layers of the TILES toolkit while *chapter 2* briefly introduced the different layers, and explained the communication protocols between them. In this subchapter we revisit the architecture and extend it with additional components in order to accommodate the requirements specified in *chapter 4*. For some of the requirements it is enough to adjust

the implementation of the existing components, while some requirements demand completely new components to be added to the system. In *Figure 5-2* we can see the architecture of the extended TILES toolkit. In this figure, the TDS component is not shown as it will be integrated into the *Web portal* component. The TADP and TEP are also not visible on this figure as they are not architectural components, but rather conceptual processes employed during development with the TILES toolkit. The following subchapter will map the requirements to the architectural components, while the implementation of the requirements is explained in *chapter 6*.

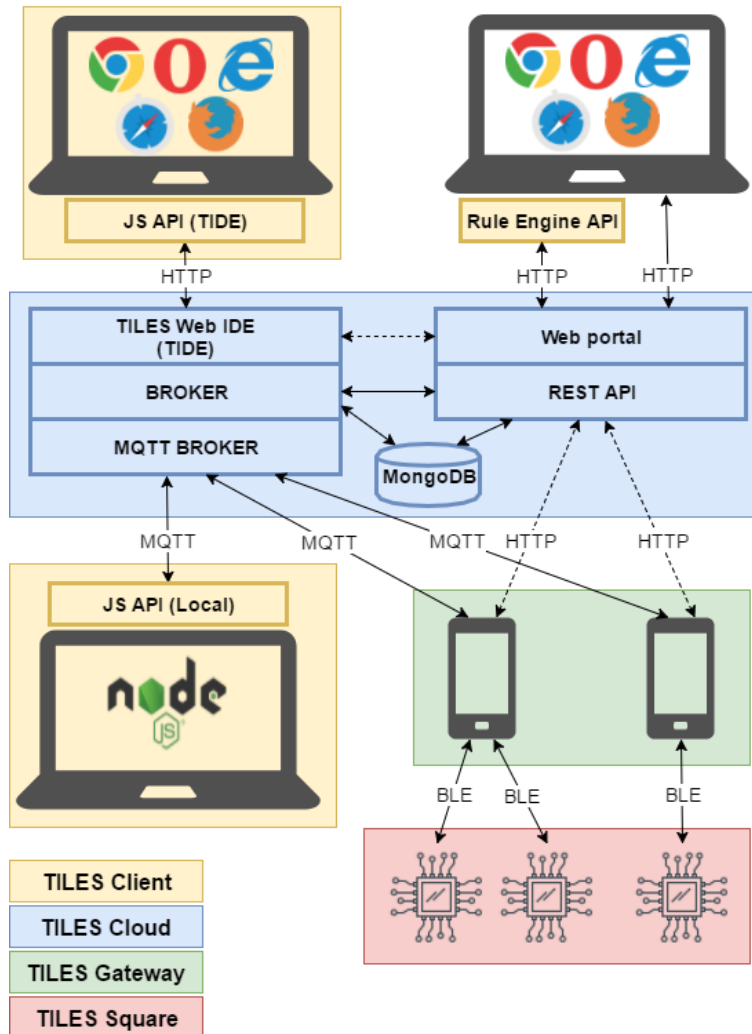


Figure 5-2, Extended TILES toolkit system architecture

5.2.1 Requirement allocation

Before starting the implementation of the extended TILES toolkit, the requirements, specified in *chapter 4*, should be allocated to the architectural elements of the system. *Figure 5-3* illustrates how the requirements are mapped to the various components of the system. In the figure we can see that some of the requirements are allocated to one specific component, while other requirements affect multiple components across several layers of the architecture. As an example, we can see that *AR13*, regarding

security in the TILES Cloud platform, is allocated to multiple components. This entails that implementing this specific requirement require a considerable amount of effort, and will affect the implementation of many components. The *IR* requirements, on the other hand, are almost exclusively allocated to a single component. This entails that implementing these requirements has less impact on the system as a whole.

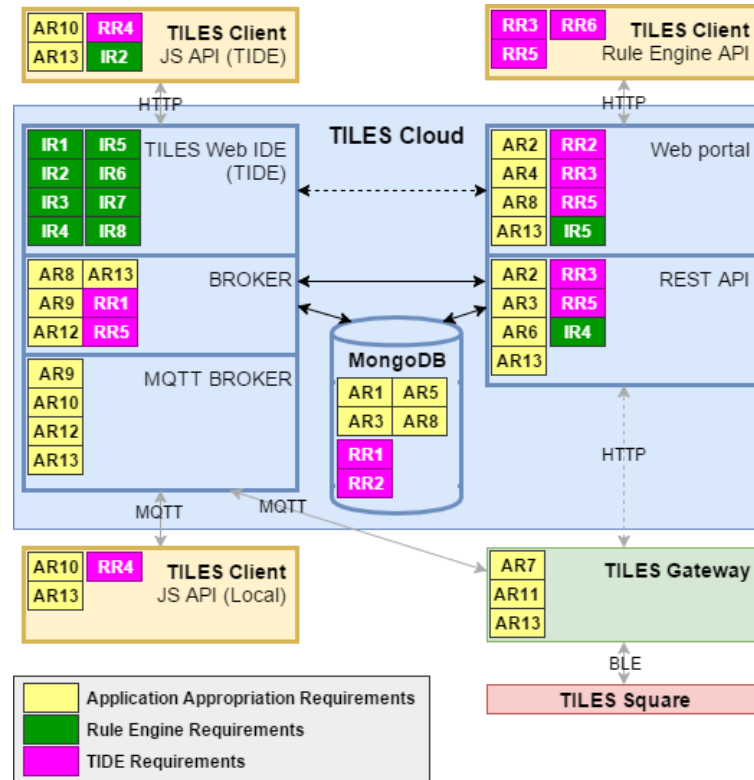


Figure 5-3, Toolkit architectural element requirement allocation

5.3 TILES Cloud web portal

Confirming the UI of the TILES Cloud web portal to the TADP and TEP is important in order to support rapid prototyping of IoT applications. The TILES Cloud web portal is the main entry point of every project as it is used to create and configure applications and select and launch development environments, in addition to accommodate the TDS. The URS, introduced in *chapter 4.4*, provide guidelines for the development of the design of the new TILES Cloud web portal. *UR6* is one that is very specific as it specifies that twitter's¹² Bootstrap¹³ framework should be employed for styling the web portal. According to their website, Bootstrap is the most popular HTML, CSS and JS framework for developing responsive projects on the web. It is a free and open-source front-end web framework for designing websites, it features standard design elements and is easily customizable to fit most projects, which makes Bootstrap a perfect candidate to ensure a familiar user experience on the TILES Cloud web portal.

¹² <https://twitter.com/>

¹³ <http://getbootstrap.com/>

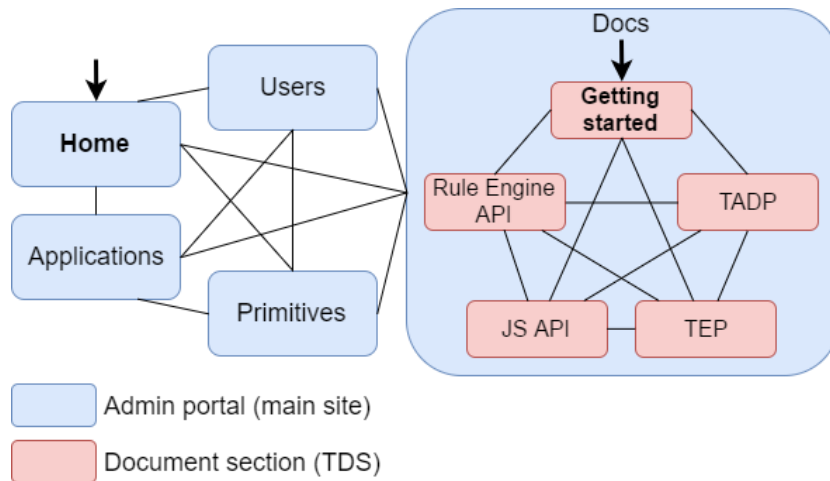


Figure 5-4, TILES Cloud web portal page graph

In *Figure 5-4*, we see an illustration of the graph of the navigation pattern of the new TILES Cloud web portal. The main entry point of the website is the *home* page and from here we will be able to navigate directly to each main section of the website. The *docs* section has, as specified by the URS, a separate navigation section enabling the user to freely navigate within this section. By looking at the available tools in Bootstrap and analyzing several online documentation sections, the new design of the TILES Cloud web portal was developed. It uses the logo and colors from the homepage of the TILES toolkit¹⁴ to create a familiar appearance across the websites of the TILES Project. The design of the main navigation panel can be seen in *Figure 5-5*. The navigation panel of the document section will be seen as a vertical navigation panel, placed on the left side of the website when the *Docs* tab is active.

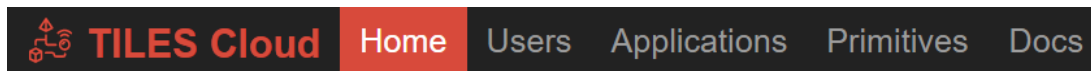


Figure 5-5, TILES Cloud web portal main navigation panel

¹⁴ <http://tilestoolkit.io>

6 Implementation

A model of the extended TILES toolkit with all its components was illustrated in *Figure 5-2* and the mapping between the requirements for the new system and the architectural components can be seen in *Figure 5-3*. This chapter will explain the technical implementation of each component, and describe how these components fit together to form the new extended TILES toolkit. The implementation explained in this section was the main part of the *Design Cycle* of the *Design Science Research*, seen in *Figure 1-2*, and was carried out as an iterative process with short *Design Cycles*. The evaluation carried out in this project and detailed in *chapter 7* helped test the implementation. All implementation described in this chapter was carried out on the TILES Cloud.

6.1 Application Appropriation

In order to conform the TILES toolkit to the functional requirements for the application appropriation listed in *Table 4-2*, the implementation of the MQTT broker and the backend functionality of the TILES toolkit had to be modified. The following subchapters will detail some of the most important changes to the implementation, with a requirement justification summary in *chapter 6.1.4*.

6.1.1 TilesAscoltatore

As described in *chapter 2.2.1.3*, the application topics for publish and subscribe are constructed from the unique identifier of the TILES Squares and the username of the owner of the Square. This means that there is no way to differentiate between different applications in the topics of the broker. To enable clients and gateways to publish and subscribe to specific application topics, the core implementation of the broker, or *TilesAscoltatore*, residing in the *tiles_ascoltatore.js* file, had to be modified. Introducing the application as a parameter in the topics is necessary to support the *Application Appropriation Requirements* and to allow the applications to run in sandbox mode without interfering with each other.

In *Code snippet 6-1* we can see part of the modified *TilesAscoltatore*, illustrating how a topic is added to the *matcher* of the broker. The matcher will look for topics that

matches the string `'tiles/evt/+/+/+/'`, where `'+'` is a wildcard matching any string. When a matching topic is detected, the broker will split the topic to find the *username*, *appid* and *deviceId* as seen on *lines 7-10* in the code snippet. This illustrates how the application name is added to the topic as *appid* and how the broker will parse the topic at runtime. On *line 13* of the code snippet, we see that the *TilesApi* is called with the *deviceId* of the TILES Square, the *username* of the user, and the *appid* of the application. This will pass the message received together with the broken down topic to the *TilesApi* for further processing the message. With this implementation, the broker will be able to detect the target application of the topic, which enables application specific logic to run, together with publishing and subscribing to events of specific applications.

```

1  function TilesAscoltatore(settings) {
2
3    ...
4
5    // Application context
6    this._matcher.add('tiles/evt/+/+/+/', function (topic,
7      message, options) {
8      var splitTopic = topic.split('/');
9      var username = splitTopic[2];
10     var appid = splitTopic[3];
11     var deviceId = splitTopic[4];
12     if (deviceId !== 'name' && deviceId !== 'active') {
13       var state = arrayBufferToString(message);
14       TilesApi.setDeviceState(deviceId, username, appid, state,
15         null);
16     }
17   });
18   ...
19   this.emit("ready");
20 }

```

Code snippet 6-1, TilesAscoltatore application topic subscription

Another important change in the broker implementation is in the unsubscribe method seen in *Code snippet 6-2*. In requirement *AR8*, it is specified that the availability state of the physical TILES Squares should be available for the users of the toolkit. This entails that when an unsubscribe event is detected, the Square state should be updated in the platform. The *deregister* call to the *TilesApi* seen in the code snippet on *line 7* represents this specific requirement. Here the topic is passed to the *TilesApi* for further processing when an unsubscribe message is received, which will make sure that the state of the Squares are updated to match the actual availability. The availability of the Squares will then be presented to the users through the TILES Cloud web portal.

```
1 TilesAscoltatore.prototype.unsubscribe = function
  unsubscribe(topic, callback, done) {
2   this._raiseIfClosed();
3
4   debug("deregistered subscriber for topic " + topic);
5   console.log(tag + "Deregistered subscriber for topic '" + topic
  + "'");
6
7   TilesApi.deregister(topic);
8
9   this._matcher.remove(topic, callback);
10  defer(done);
11 };
```

Code snippet 6-2, TilesAscoltatore prototype unsubscribe

6.1.2 TilesApi

The *TilesApi*, implemented in the file *tiles_api.js*, exist to support device specific management upon incoming events and commands. When an incoming event with the proper topic structure is detected by the broker, as seen in *Code snippet 6-1*, the *setDeviceState* of the *TilesApi* will be called as seen in *line 13* in the same code snippet. This method will be responsible for updating the state of the TILES Squares in the database by calling the appropriate service from the REST API. The *TilesApi* had to be modified to support the application identifier of a topic, as well as to accommodate the availability state and last received message state with a timestamp in the model. It is the *TilesApi* that holds the responsibility to act upon the incoming events and commands that the broker detects.

6.1.3 MongoDB

In order to support the application appropriation and all its requirements, the MongoDB database had to be updated with new models for the application context and abstract placeholders in the database. For this purpose, two new models were added to the database, *VirtualTiles* to represent the abstract placeholders, seen in *Code snippet 6-3*, and *Applications* to represent the application context, seen in *Code snippet 6-4*. These two models were added without changing any of the existing models in order to minimize the impact on the whole system. This entails that there is no reference from the *users* in the database to the *applications*. Having a reference from the user to their configured applications would be useful, but this would require changes to the existing database models, which would require additional effort on updating the infrastructure of the system. Instead the applications will hold a reference to the application owner, which will enable getting a list of a user's application by checking the *Applications* model in the database. For the *VirtualTiles* however, there is a two way reference. The *VirtualTiles* will keep a reference to the *application* to which they belong, and the *applications* will keep a reference to all the *VirtualTiles* defined in their scope as seen in the code snippets below.

```

1  var mongoose = require('mongoose');
2
3  var VirtualTileSchema = new mongoose.Schema({
4    virtualName: String,
5    tile: { type: String, ref: 'Tile' },
6    application: { type: String, ref: 'Application' }
7  });
8
9  mongoose.model('VirtualTile', VirtualTileSchema);

```

Code snippet 6-3, VirtualTiles MongoDB databasemodel

```

1  var mongoose = require('mongoose');
2
3  var ApplicationSchema = new mongoose.Schema({
4    _id: String,
5    devEnvironment: String,
6    environmentOnline: Boolean,
7    appOnline: Boolean,
8    port: Number,
9    iftttkey: String,
10   user: { type: String, ref: 'User' },
11   virtualTiles: [{ type: mongoose.Schema.Types.ObjectId, ref:
12     'VirtualTile' }]
13 });
14 ApplicationSchema.methods.addVirtualTile = function(vt, cb) {
15   this.virtualTiles.addToSet(vt);
16   this.save(cb);
17 }
18
19 mongoose.model('Application', ApplicationSchema);

```

Code snippet 6-4, Applications MongoDB databasemodel

6.1.4 Requirement Rationale

In this subchapter, all the *Application Appropriation Requirements*, introduced in *chapter 4.1*, will be mapped to the files implementing the functionality. The tables in the following subchapters list what files were added and what files were modified to accommodate the individual requirements.

6.1.4.1 AR1

The TILES Cloud platform should support creating, listing, configuring and deleting applications (CRUD operations on applications).

To accommodate AR1, a new database object model had to be added to the database. This model can be seen in *Code snippet 6-4*.

Modified files	- <i>app.js</i>
New files	- <i>models/Applications.js</i>

Table 6-1, AR1 implementation files

6.1.4.2 AR2

The CRUD operations should be available through REST services and the TILES Cloud web portal.

This requirement specifies availability through REST services, thus the REST API had to be modified by introducing a new file for application REST services. In addition, two new pages were added to the web portal, thus two new template files were added to the views directory, together with modifying the JavaScript files for making the CRUD operations available in the TILES Cloud web portal.

Modified files	<ul style="list-style-type: none"> - <i>app.js</i> - <i>public/javascripts/app.js</i> - <i>public/javascripts/controllers.js</i> - <i>public/javascripts/services.js</i>
New files	<ul style="list-style-type: none"> - <i>routes/applications.js</i> - <i>views/templates/applications.ejs</i> - <i>views/templates/application.ejs</i>

Table 6-2, AR2 implementation files

6.1.4.3 AR3

The TILES Cloud platform should support abstract devices or placeholders to be created and removed from TILES applications.

To accommodate AR3, the Application model in the database and the REST services for configuring applications were modified. In addition, the *VirtualTiles* model, seen in *Code snippet 6-3*, was introduced. This model represents the abstract placeholders of physical objects in an application.

Modified files	<ul style="list-style-type: none"> - <i>models/Applications.js</i> - <i>routes/applications.js</i>
New files	<ul style="list-style-type: none"> - <i>models/VirtualTiles.js</i>

Table 6-3, AR3 implementation files

6.1.4.4 AR4

Configuring abstract devices should be available through the TILES Cloud web portal.

Enabling configuration of abstract devices through the TILES Cloud web portal for AR4 required changes to several source files, including JavaScript files and view template files.

Modified files	<ul style="list-style-type: none"> - <i>app.js</i> - <i>public/javascripts/app.js</i> - <i>public/javascripts/controllers.js</i> - <i>public/javascripts/services.js</i> - <i>routes/applications.js</i> - <i>views/templates/application</i> - <i>views/templates/applications</i>
-----------------------	--

Table 6-4, AR4 implementation files

6.1.4.5 AR5

The TILES Cloud platform should support pairing abstract devices with physical TILES Squares.

Only one file had to be modified to accommodate AR5. The *VirtualTiles* database model was updated to hold a reference to the physical TILES Square as can be seen in *Code snippet 6-3*. For this requirement, the reference to the physical TILES Square was added, which would enable the *TilesApi* to map between the abstract and physical TILES Square at runtime by looking up the abstract Square in the database.

Modified files	- <i>models/VirtualTiles.js</i>
-----------------------	---------------------------------

Table 6-5, AR5 implementation files

6.1.4.6 AR6

Pairing abstract and physical TILES Squares should be available through REST services at application runtime.

To enable pairing of abstract and physical TILES Squares through REST services, only the applications REST API file had to be modified.

Modified files	- <i>routes/applications.js</i>
-----------------------	---------------------------------

Table 6-6, AR6 implementation files

6.1.4.7 AR7

The TILES Gateway should be responsible for pairing the discovered physical TILES Squares with the abstract devices.

To accommodate AR7, the modifications already implemented for AR5 and AR6 are sufficient from server side. In addition, the implementation of the TILES Gateway had to be modified. Due to the complexity of the modifications for the TILES Cloud server, the required modifications of the TILES Gateway app was outsourced [1].

6.1.4.8 AR8

The last events and availability state of physical devices paired with an abstract devices should be accessible in the TILES Cloud web portal.

To accommodate AR8, the first thing that had to be modified was the *Tiles* database model so that it would store the latest received event. In addition, the REST API was updated with an additional method for storing the events of the physical device. Last, the broker and TILES API implementation was modified to pass the last event to the database upon incoming events. Due to the implementation of AR5, the *TilesApi* is already able to look up the physical TILES Square referenced by the abstract Square in the *VirtualTiles* database model.

Modified files	- <i>models/Tiles.js</i> - <i>tiles_api.js</i> - <i>tiles_ascoltatore.js</i> - <i>routes/index.js</i>
-----------------------	--

Table 6-7, AR8 implementation files

6.1.4.9 AR9

The TILES Cloud platform should support publishing and subscribing to TILES Squares using the paired abstract squares of an application.

To accommodate AR9 only the implementation of the broker had to be modified. Publish and subscribe methods were modified to support publish and subscribe to the abstract squares.

Modified files	- <i>tiles_ascoltatore.js</i>
-----------------------	-------------------------------

Table 6-8, AR9 implementation files

6.1.4.10 AR10

The TILES Cloud platform should support using the abstract devices in application development without concern for the physical hardware devices.

Similarly to AR9, only the broker had to be modified to accommodate AR10. Even though only one file was modified, this is a very important piece of implementation as it enables the name of the *abstract placeholder* to be used in TILES application program code instead of the hardcoded unique identifier of the physical TILES Squares.

Modified files	- <i>tiles_ascoltatore.js</i>
-----------------------	-------------------------------

Table 6-9, AR10 implementation files

6.1.4.11 AR11

The TILES Gateway should support user actions such as selecting and running applications.

To accommodate AR11, the REST services had to be modified. The gateway will call the REST API in order to communicate with the background services of the TILES Cloud server. In addition, the TILES Gateway implementation had to be modified [1].

Modified files	- routes/applications.js
-----------------------	--------------------------

Table 6-10, AR11 implementation files

6.1.4.12 AR12

The TILES Cloud platform should support pairing abstract devices with third party embedded hardware devices.

Due to time constraints of the project, this requirement was discarded and is suggested for future research, seen in *chapter 9.3*. Support for third party embedded hardware devices is a nice-to-have feature in the TILES toolkit, but with the custom ports of the TILES Squares most scenarios are already supported by the TILES Squares hardware devices.

6.1.4.13 AR13

The TILES Cloud platform should implement modern security mechanisms to protect the users' applications from unauthorized access.

Accommodating AR13 has a significant impact on the whole system as seen in *Figure 5-3*. In addition it requires a significant amount of effort and time, thus this requirement was discarded. The protocols used in the TILES toolkit (MQTT, HTTPS) does support security mechanisms to be implemented, thus this requirement is suggested as a future research opportunity, as seen in *chapter 9.3*.

6.2 Rule Engine Environment

Conforming the TILES toolkit to the functional requirements for the Rule Engine Development Environment, listed in *Table 4-3*, the web portal, the REST API and the client APIs had to be modified as seen in *Figure 5-3*. The following subchapters will detail some of the most important implementation for these requirements, followed by a requirement justification summary in *chapter 6.2.3*.

6.2.1 MongoDB

Several of the requirements for the Rule Engine requires new database models to be introduced to the toolkit. To accommodate these requirements, three new database models were added. The *Tilehooks* model, seen in *Code snippet 6-5*, was added to store the rules defined in a Rule Engine application. In the code snippet we can see that the *Tilehooks* model stores a reference to the *Application*, the input *VirtualTile* and the output *VirtualTile*. Upon incoming events the broker will call the *TilesApi* which in term will look up the rules belonging to the specific application and input TILES Square. If a rule has been defined, the *TilesApi* will trigger the command defined in the *Tilehook* and send it to the registered output TILES Square.

```
1 var mongoose = require('mongoose');
2
3 var TilehooksSchema = new mongoose.Schema({
4   application: { type: String, ref: 'Application' },
5
6   //Input Tile and properties
7   virtualTile: { type: mongoose.Schema.Types.ObjectId, ref:
8     'VirtualTile' },
9   tigger: String,
10  properties: [String],
11
12  //Output Tile and properties
13  outputVirtualTile: { type: mongoose.Schema.Types.ObjectId,
14    ref: 'VirtualTile' },
15  outputTrigger: String,
16  outputProperties: [String]
17 });
18 mongoose.model('Tilehook', TilehooksSchema);
```

Code snippet 6-5, Tilehooks, MongoDB database model

The *Iftthooks* model, seen in *Code snippet 6-6*, was added to store the IFTTT rules defined in a Rule Engine application. The *Iftthooks* model stores a reference to a *VirtualTile*, which is the abstract TILES that the rule belongs to. An IFTTT rule can either be outgoing or incoming. If the rule is outgoing, it means that upon matching the incoming event from the TILES Square, a POST request will be triggered to the registered IFTTT applet. This database model will store all rules involving IFTTT applets in a Rule Engine application. If the rule is incoming, it means that the system will be listening to incoming POST request, and the stored IFTTT rule will be triggered to send a command to appropriate TILES Square.

```

1  var mongoose = require('mongoose');
2
3  var IfttthooksSchema = new mongoose.Schema({
4    triggerName: String,
5    outgoing: Boolean,
6    trigger: String,
7    properties: [String],
8    virtualTile: { type: mongoose.Schema.Types.ObjectId, ref:
9      'VirtualTile' },
10   application: { type: String, ref: 'Application' },
11 });
12 IfttthooksSchema.methods.getPostUrl = function () {
13   if (!this.application.iftttkey) return '';
14   return "https://maker.ifttt.com/trigger/" + this.triggerName
15     + "/with/key/" + this.application.iftttkey;
16 }
17 mongoose.model('Ifttthook', IfttthooksSchema);

```

Code snippet 6-6, Ifttthooks, MongoDB database model

The *Primitives* model, seen in *Code snippet 6-7*, was added to store all the input and output primitives available in a Rule Engine application. If the firmware of the TILES Squares is updated with new data primitives, these primitives will not be available in a Rule Engine application automatically. By registering the primitives to the database, the Rule Engine will be able to dynamically update the list of available operations in the Rule Engine environment, which will simplify the implementation of new primitives considerably. The model stores the *name* and *properties* fields, as these are needed to match against the messages exchanged with the TILES Squares. A Boolean flag is stored to detect if the primitive is an *output* or *input* primitive.

```

1  var mongoose = require('mongoose');
2
3  var PrimitiveSchema = new mongoose.Schema({
4    isInputPrimitive: Boolean,
5    name: String,
6    properties: [String],
7    hasCustomProp: Boolean
8  });
9
10 mongoose.model('Primitive', PrimitiveSchema);

```

Code snippet 6-7, Primitives, MongoDB database model

6.2.2 REST API

To enable the *Tilehooks* and *Ifttthooks* to be configured through REST services and the TILES Cloud web portal, new services were added for creating, reading and deleting the hooks in the REST API. In *Code snippet 6-8* we can see the implementation of the POST service that will create a new *Tilehook* in the database. The *Tilehook* model will be passed in the body of the request as seen in *line 1*. The *Tilehook* will be created and saved to the database with a reference to the *VirtualTile*

model. The service will return the newly created hook as a JSON object. Similar methods exist for creating, reading and deleting *Tilehooks*, *Iftthooks* and *Primitives*.

```

1  router.post('/:app', function (req, res, next) {
2    var data = req.body;
3    data.application = req.params.app;
4
5    var tilehook = new Tilehook(data);
6
7    tilehook.save(function (err, tile) {
8      if (err) return next(err);
9      Tilehook.populate(tile, [{ path: "virtualTile" },
10     { path: "outputVirtualTile" },
11     { path: "application" }], function (err, tilenew) {
12       if (err) return next(err);
13       res.json(tilenew);
14     });
15   });
16 });

```

Code snippet 6-8, REST API, POST service, Create new Tilehook

6.2.3 Requirement Rationale

Revisiting the *Rule Engine Requirements* introduced in *chapter 4.2*, this subchapter will map the requirements to the files implementing the functionality. The tables in the following subchapters list what files were added, and what files were modified to accommodate the individual requirements.

6.2.3.1 RR1

The TILES Cloud platform should support application development by defining rules without having to write code textually.

To store the rules, a new database model had to be created. This is the *Tilehooks* database model seen in *Code snippet 6-5*. In addition the *TilesApi* and *broker* (*TilesAscoltatore*), had to be modified to check all incoming events to see if a rule has been defined on the TILES Square. This way there is no textual program code running for the application, only rules stored in the database that will be looked up on incoming events.

Modified files	- <i>app.js</i> - <i>tiles_api.js</i> - <i>tiles_ascoltatore.js</i>
New files	- <i>models/Tilehooks.js</i>

Table 6-11, RR1 implementation files

6.2.3.2 RR2

The TILES Cloud platform should provide a Rule Engine API that enables expert-users to maintain a list of available TILES rules to be used by non-experts.

To maintain a list of available primitives, the *Primitives* database model, seen in *Code snippet 6-7*, was introduced together with new REST services for creating, reading and deleting primitives in the database. Additionally, a new view was added in the TILES Cloud web portal to enable experts to maintain the list of primitives.

Modified files	- <i>app.js</i>
New files	- <i>models/Primitives.js</i> - <i>routes/primitives.js</i> - <i>views/templates/primitives.ejs</i>

Table 6-12, RR2 implementation files

6.2.3.3 RR3

The TILES Cloud platform should support creating, configuring and deleting rules through REST services and the TILES Cloud web portal.

Since the rules of the Rule Engine applications should be available through both REST services and the web portal, the application view and the REST API had to be modified. There was previously no services for managing *Tilehooks* in the REST API, thus this file had to be created.

Modified files	- <i>app.js</i> - <i>views/templates/application.ejs</i>
New files	- <i>routes/tilehooks.js</i>

Table 6-13, RR3 implementation files

6.2.3.4 RR4

The TILES Cloud platform must keep the support for existing development environments upon introduction of the new rule based environment.

In order to keep support for the existing development environments, the implementation of the Rule Engine had to be carefully constructed not to modify any existing database model or REST API services that would break the support for the textual JavaScript client development.

6.2.3.5 RR5

The TILES Cloud platform should offer a rule based approach for integrating the application with IFTTT without having to write code textually.

To accommodate this requirement, the *Ifttthooks* database model, seen in *Code snippet 6-6*, was introduced, together with the implementation of the REST services for creating, reading and deleting the rules. In addition, the *TilesApi* was modified to check the database for *IFTTT* rules together with the *Tilehooks* rules, whenever the broker detected an incoming event.

Modified files	- <i>app.js</i> - <i>tiles_api.js</i>
New files	- <i>models/Ifttthooks.js</i> - <i>routes/ifttthooks.js</i>

Table 6-14, RR5 implementation files

6.2.3.6 RR6

The Rule Engine should support the ability to automatically generate JavaScript code using the JS APIs for every Rule Engine application.

Being able to automatically generate JavaScript code based on a Rule Engine application is considered to be a great tool for non-experts that will help them get started with the TILES toolkit, and teach them the syntax of JavaScript and how the TILES JavaScript APIs are working. However, due to time constraints and the low priority of the requirement, RR6 has been temporarily discarded and is suggested as a future research opportunity, as seen in *chapter 9.3*.

6.3 TILES toolkit IDE

Most of the implementation for conforming the TILES toolkit to the functional requirements of the TILES toolkit IDE, listed in *Table 4-4*, resides in a single architectural component, seen in *Figure 5-3*. This entails that introducing the functional requirements of TIDE should have little impact on other system functionalities. The following subchapters will detail the most important implementation for these requirements, followed by a requirement justification summary in *chapter 6.3.3*.

6.3.1 Cloud9

As seen in *Figure 5-3*, almost all the requirements for the TIDE resides in a single architectural components. To realize these functional requirements, this new component had to be introduced into the TILES toolkit. Instead of implementing an IDE from scratch, it was decided that integrating a third party IDE should suffice.

Derived from the IR requirements, the following requirements was constructed for the third party IDE:

- *IDE should run the development environment in a browser*
- *IDE should provide a textual editor for writing JavaScript code with Node.js*
- *IDE should support running multiple instance of the environment*
- *IDE should support custom configurable workspaces on the host server*
- *IDE should support debugging applications in the web browser*

Cloud9¹⁵ is an open source web IDE that supports hundreds of programming languages, including JavaScript with Node.js. It runs a development environment in the browser, it can be configured with custom workspaces, it has debugging capabilities, and with only some effort it can run multiple instances of the environment, thus it fulfills all the requirements listed above. This makes Cloud9 a suitable candidate for TIDE in the TILES toolkit.

Appendix A shows instructions on how to set up Cloud9 as TIDE on the TILES Cloud server. In addition to download and configure Cloud9 to run on the TILES Cloud server, configuring workspaces and starting and stopping the independent instances of the environment had to be integrated into the TILES toolkit in order to make TIDE an effortlessly extension of the TILES toolkit to be used by developers. For this purpose, several helper functions were implemented, which will be detailed in the following subchapter.

6.3.2 Background processes

As described in the previous subchapter, the Cloud9 IDE supports several of the functional requirements for TIDE. However, in order to transform the Cloud9 IDE from a third party standalone service into TIDE, and make it an integrated part of the TILES toolkit, some background operations on the TILES Cloud server had to be modified. Most notably, when a TIDE application is created in the web portal, the TILES Cloud server should set up the workspace with the TILES JavaScript API automatically. In addition, launching an instance of TIDE for a specific application workspace should be possible without knowledge of the Cloud9 framework. Supporting these operations in the backend is necessary in order make usage of the TIDE an effortless activity for non-experts employing the TADP. To achieve this, some helper functions had to be added to the TILES toolkit that will be triggered when an application is created.

In *Code snippet 6-9* we can see the *createWorkspace* helper function implemented for configuring the workspace for an application in TILES Cloud. This method is called when a TIDE application is created, and will be responsible for setting up the workspace and configuring the template files in the workspace. On *line 5* we can see that a native Linux command is executed on the server, which will create a new workspace in the workspace root directory. If this is created successfully, the code on

¹⁵ <https://c9.io/>

line 7 will be executed, which will copy the JavaScript API template files from the template root directory defined in the *config* file, into the workspace directory before *renameApp* will be called to rename the application name of the template to the current application name.

```
1 var createWorkspace = function (workspace, username) {
2   var tag = "[ERROR Creating workspace] ";
3
4   if (process.platform === "linux") {
5     exec("sudo -H -u c9sdk bash -c 'mkdir " +
6         config.cloud9.workspace.root + workspace + "'", function
7         (error) {
8       if (error) return;
9       exec("sudo -H -u c9sdk bash -c 'cp " + config.lib.root +
10          "/templates/* " + config.cloud9.workspace.root + workspace
11          + "'", renameApp);
12    });
13  }
14  else {
15    console.log(tag + "Cloud workspace only available on linux");
16  }
17 }
```

Code snippet 6-9, Create Workspace helper function

A short description of all the helper functions for TIDE can be seen in the following subchapter under the relevant requirement section.

6.3.3 Requirement Rationale

Revisiting the *TILES toolkit IDE Requirements* introduced in *chapter 4.3*, this subchapter will describe and justify the implementation of the individual requirements and list what files were added or modified during the implementation.

6.3.3.1 IR1

The TIDE should run in a web browser, and be hosted by the TILES Cloud.

This requirement is supported by integrating the third party Cloud9 framework into the TILES Cloud server. No source code files were added or modified. See *Appendix A* for instructions on how the Cloud9 environment was configured.

6.3.3.2 IR2

The TIDE should provide a textual editor for writing JavaScript code.

To accommodate this requirement, two helper functions were implemented in the applications REST API and a button for triggering the functions were added to the application view in the web portal. Triggering these functions requires explicit user action in the web portal.

<i>startHostingWorkspace</i>	Will start a new instance of the TIDE for the specific application and give the application a URL to the hosted TIDE. Called when <i>start hosting</i> is selected in the TILES Cloud web portal.
<i>stopHostingWorkspace</i>	Will stop the TIDE instance running for the specific application. Called when <i>stop hosting</i> is selected in the TILES Cloud web portal.

Modified files	- <i>routes/applications.js</i> - <i>views/templates/application.ejs</i>
-----------------------	---

Table 6-15, IR3 implementation files

6.3.3.3 IR3

The TILES Cloud platform should support creating TIDE workspaces.

To accommodate this requirement, the *createWorkspace* and *removeWorkspace* helper functions were implemented. The implementation of these resides in the applications REST API and will be called when applications are created or deleted in the TILES Cloud web portal. Both functions are called in the background without being explicitly initiated by the users.

<i>createWorkspace</i>	Will create workspace directory, copy JavaScript API template files and replace setup in template files with application specific details, seen in <i>Code snippet 6-9</i> . Called when a new application is created in TILES Cloud web portal.
<i>removeWorkspace</i>	Will take down the application workspace by removing the directory and template files (not reversible). Called when an application is deleted in the TILES Cloud web portal.

Modified files	- <i>routes/applications.js</i>
-----------------------	---------------------------------

Table 6-16, IR3 implementation files

6.3.3.4 IR4

The TILES Cloud platform should automatically configure the workspace with the JavaScript API for all TIDE applications.

The *createWorkspace* and *removeWorkspace* helper functions described for IR3 were slightly modified to accommodate this requirement. The new implementation will make sure that the workspace is properly configured with the JavaScript API, in addition to configure the template files for the target application. The helper functions are still initiated by creating and removing an application, and will not need to be explicitly initiated by the user.

Modified files	- <i>routes/applications.js</i>
-----------------------	---------------------------------

Table 6-17, IR4 implementation files

6.3.3.5 IR5

The TILES Cloud platform should support deployment of TIDE applications by running the JavaScript client code directly from the workspace location in the Cloud.

For this requirement, two additional helper functions were implemented in the applications REST API, and a button for initiating the functionality was added to the web portal application view. These helper functions are explicitly initiated by user actions and will be called by the REST API.

<i>startApplication</i>	Will start the application program code, implemented in <i>tiles.js</i> template file in workspace, as a service on TILES Cloud, enabling it to run in production mode. Called when <i>start application</i> is selected in the TILES Cloud web portal.
<i>stopApplication</i>	Will stop the service running the application code in production mode. Called when <i>stop application</i> is selected in the TILES Cloud web portal.

Modified files	- <i>routes/applications.js</i> - <i>views/templates/application.ejs</i>
-----------------------	---

Table 6-18, IR5 implementation files

6.3.3.6 IR6

The TIDE should support debugging applications and logging output to a console window in the browser.

Support for this requirement is inherited by using Cloud9 as TIDE. Cloud9 has a debugging feature allowing users to run Node.js applications directly in the environment.

6.3.3.7 IR7

The TILES Cloud platform should support automatic JS code template generation for the abstract devices configured in the application context.

Adding automatic code generation upon adding and removing abstract devices in an application was solved by implementing two additional helper functions. These helper functions are not explicitly initiated by the user, but will be called in the background when abstract TILES Square placeholders are added or removed from an application. The helper functions will open the JavaScript templates and add the code for initializing the abstract TILES Square in the code.

<i>addVtToTemplate</i>	Will add JavaScript template for an abstract TILES Square in the workspace template files. Called when an abstract TILE is added to an application.
<i>removeVtFromTemplate</i>	Will remove the JavaScript template code for an abstract TILES Square in the workspace template files. Called when an abstract TILE is removed from an application.

Modified files	- <i>routes/applications.js</i>
-----------------------	---------------------------------

Table 6-19, IR7 implementation files

6.3.3.8 IR8

The TILES Cloud platform should support security measures to protect the private TIDE application workspaces from unauthorized access.

To support protecting the private instance of TIDE for an application workspace, the web view of TIDE must be protected by the service hosting the IDE. Cloud9 automatically adds support for security, but this feature has been disabled temporarily in the code as the rest of the TILES Cloud platform currently has no implemented security mechanisms. Support for this requirement, however, is supported and can easily be switched on as soon as the rest of the Cloud infrastructure is properly secured.

7 Evaluation

This chapter is dedicated to the research illustrated on *Figure 1-2, Design Science Research, Environment (adopted from [9])*. In this part of the research, I was tapping into the *Environment* through the *Relevance Cycle* of the *Design Science Research* by conducting several iteration of evaluation.

For evaluation, two workshops were conducted with non-experts to evaluate the TADP, and a focus group with expert users was organized to evaluate the TEP. In addition, the students implementing the new TILES Gateway smartphone app [1] provided constant feedback on the usability and functionality of the background services during their work with the new gateway implementation, which acted as an evaluation on the implementation of the background services. The workshops conducted for non-experts in this project builds on the workshop structure constructed in the specialization project [23].

7.1 Non-expert workshops

This subchapter will explain the workshops conducted for evaluating the tools of the extended TILES toolkit targeting non-expert users.

7.1.1 Objectives

The objective of the non-expert workshops was to test the new tools of the extended TILES toolkit focusing on the transition from the ideation phase to prototyping. More precisely the TADP, TIDE, TILES Cloud web portal and all the support documentation available to the non-experts during the workshop in TDS were evaluated. The GUI and UX of the web portal was not the main focus of evaluation, however some fundamental improvements of the GUI were necessary in order to support rapid application prototyping as per the research questions of this thesis.

The TADP introduced in *chapter 3* and the *functional requirements* introduced *chapter 4* will act as the base evaluation criteria for the non-expert evaluation. The hypothesis to be tested during the non-expert evaluation can be seen below. The main difference between these workshops and the workshops of the specialization project is the new

processes and the support documentation section enabling the non-experts to perform the workshop with no prior setup of the workshop, and minimal supervision from toolkit experts in transitioning from ideation to prototyping.

Hypothesis:

People with little programming experience should be able to prototype their ideas generated with the TILES Cards (or simple ideas generated for them) using the TILES toolkit with minimal supervision from toolkit experts, and only a brief introduction to the concept of IoT from IoT experts.

7.1.2 Research tools

The research tools employed during the workshops were *observation*, *survey* and *discussion*. The participating non-experts were given questionnaires to provide feedback on specific topics of interest after the workshop, in addition to being observed by the workshop supervisors during the workshop. The observation provided objective feedback during the workshop, while the survey and discussion enabled the non-experts to subjectively share their personal thoughts and experience.

The survey was constructed as a questionnaire, using a five-point Likert scale with 1 being the negative end of the scale (strongly disagree) and 5 being the positive end (strongly agree). The questionnaire also contained text questions where the non-experts could write a few lines of text for elaborating their answers. The questions of the questionnaire can be seen in *Table 7-1*. The letter in the rightmost column indicates whether the question was answered by the Likert scale (L) or Text input (T). A summary of the feedback is available in *chapter 7.1.5*.

	Type
Regarding the documentation	
1. The provided developer's documentation was useful during the prototyping phase	L
2. The provided developer's documentation was clear and easy to understand	L
3. The provided developer's documentation made it easy to follow the process of transitioning from ideation to application prototype	L
4. The provided developer's documentation and examples made it easy to translate the TILES Cards into JavaScript code	L
Regarding the prototyping process	
5. The steps of the prototyping process were easy to follow	L
6. I faced few challenges with the process description	L
7. If you faced any challenges, what were they?	T
8. During the prototyping process it was always clear to me what I was supposed to do	L
9. Following the steps of the prototyping process was fun	L
Regarding the scenarios	
10. The scenario(s) given for prototyping were clear and easy to understand	L
11. The scenario(s) given for prototyping were easy to implement	L
12. I recognize the need for the scenario(s) given for prototyping (I would use these applications at home)	L
Regarding the tools	
13. Navigating the TILES Cloud website was easy	L
14. Using the TILES Cloud web IDE was easy	L
15. I faced few challenges using the TILES Cloud web IDE	L
16. If you faced any challenges, what were they?	T
General	
17. Using JavaScript for prototyping TILES applications was easy	L
18. I wish I had TILES Squares at home so I could use the TILES toolkit for personal TILES applications at home	L
Please answer with a few sentences	
19. What was most difficult in the workshop?	T
20. What was easiest in the workshop?	T
21. Do you have any suggestions for improving the workshop?	T
22. Do you have any suggestions for improving the provided developer's documentation?	T
23. If you could change the TILES Cloud web IDE, what would you change?	T

Table 7-1, Non-expert survey questions

7.1.3 Participants

The participants of the first pilot workshop were five computer science students with some programming experience that were divided into two groups. These students are more experienced than the target non-expert users of the TADP, but as the structure of the workshop and syntax of APIs was tested in the specialization project [23], it was decided that running an initial pilot workshop with more experienced user was sufficient for testing the process and process description available in the TDS. In addition, being more experienced, the users were able to provide a more reflected feedback on the workshop and support documentation, and compare it to other documentation they have encountered in their previous projects. The participants already had a general understanding of the concept of IoT and they had all prior experience with JavaScript, although none of them proclaimed to be experts in any of the concepts. This made them good candidates for the initial workshop.

The participants of the second workshop were 14 students of age 15/16, which were divided into four groups. These participants were closer to the target non-experts in level of experience as they had very little knowledge of development and IoT. Most of the participants had some experience with python, but none of them had any experience with JavaScript development. Due to their age and the fact that the workshop was conducted during regular school hours, the participants were easily distracted, and their focus easily drifted, but the participants' level of experience made them good candidates for the workshop.

7.1.4 Setting up and running the workshop

The participants of the workshops should be divided into groups of two to four people, and each group should have a computer, several TILES Squares and a mobile phone running the TILES Gateway at their disposal. The computer needs a steady internet connection in order to navigate to the online documentation and the admin section on the TILES Cloud web portal. The workshop of the extended TILES toolkit is intended to be self-supporting, meaning that no prior configuration should be necessary to run the workshop. The development environment, TIDE, and support documentation, TDS, will be available through the browser running on the computer. In addition, the participants are given a predefined storyboard application, which can be seen in *Figure 7-1*. This predefined storyboard explains the application idea that the participants will prototype during the workshop.

Table 7-2 shows the workshop protocol of the non-expert evaluation. The first phase of the protocol is an introduction to IoT and the TILES Cards ideation process. This phase is not subject for evaluation in this thesis as the ideation process has already been evaluated [19] [22] [23] [30]. This phase is still necessary as the rest of the phases requires the non-experts to be familiar with the ideation process of the TILES toolkit in order to perform the transition from ideation to prototyping. The second phase is an introduction to the prototyping process where workshop supervisors will explain how to get started with the toolkit, introduce the document section on the TILES Cloud web

portal, demonstrate a few TILES application scenarios and introduce the storyboards that the non-experts will prototype in the next phase of the workshop.

The third phase of the workshop is where the non-experts will be using the introduced documentation and the provided TILES Cards storyboard to prototype the application by going through the steps of the TADP. In this phase the non-experts are encouraged to acquire the necessary information on their own from the documentation section with minimal help from workshop supervisors. The predefined storyboard application can be seen in *Figure 7-1*. In the fourth and final phase, the non-experts will be asked to fill the questionnaire, introduced in *chapter 7.1.2*, which concludes the workshop.

"Have you ever been at a party where your shoes have been separated and you can only find one of them when you are leaving? To solve this problem you should implement an application for finding the other shoe when you have located one of them."

- Double Tap on one shoe will set LEDs on both shoes and vibrate the OTHER shoe. (Should work in both ways, see example below)
 - Tap left shoe > vibrate right shoe and set LED on both shoes
 - Tap right shoe > vibrate left shoe and set LED on both shoes
- Tilt any shoe and LEDs go off on both shoes

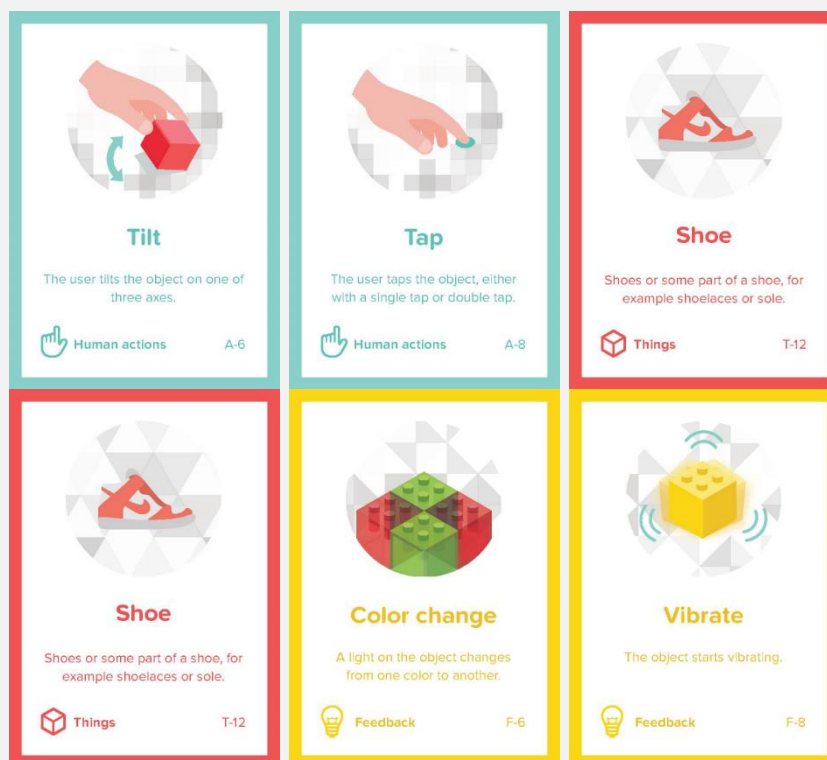


Figure 7-1, Workshop application storyboard

The predefined storyboard application seen in *Figure 7-1*, was designed prior to the workshop with support for available hardware in order to ensure that the generated idea is supported by the TILES toolkit. The storyboard was constructed to resemble applications previously designed by non-experts, but simplified and tailor to fit the

current scope of the workshop. The workshop structure even supports letting the participants prototype their own application ideas, but special emphasis must be made during the ideation phase to enable that only ideas using the supported hardware and software capabilities are used in the ideation phase.

Time	What	Why
<i>First part: INTRO to IoT and IDEATION</i>		
60-120 min	<ul style="list-style-type: none"> - Intro to IoT - Intro to TILES Cards - Playing TILES Cards 	Make sure the participants understand the concept of IoT, the TILES Cards and how to use them in ideation.
<i>Second part: INTRO TO PROTOTYPING</i>		
10 min	Getting started <ul style="list-style-type: none"> - Components of TILES - Layers of architecture - TILES Squares primitives 	Provide basic knowledge of the TILES toolkit that the participants will use in their prototype later (Squares, Gateway, Cloud, Clients) <i>(this is the Getting Started section of the TDS on TILES Cloud web portal)</i>
10 min	<ul style="list-style-type: none"> - Show example scenarios with TILES Squares attached to objects 	Show one/two sample scenarios so that the participants can understand how to interact with the TILES Squares in their own applications
5 min	<ul style="list-style-type: none"> - Introduce Docs section on web portal - Explain the Application Development Process chapter on Docs 	Explain where to find information about the steps of prototyping, show the participants how to use the Docs section.
5 min	<ul style="list-style-type: none"> - Introduce predefined storyboard 	Show the participants the predefined TILES Cards storyboard that they will be prototyping
<i>Third part: DEVELOPING PROTOTYPES</i>		
30-60 min	<ul style="list-style-type: none"> - Let participants prototype the predefined IoT application - Tell the participants to navigate to the Docs section of the web portal and let them start prototyping by going through the App. Dev. Process. 	Encourage the participants to follow the steps of the process and let them try to get started without further instructions from workshop supervisors.
<i>Fourth part: EVALUATION</i>		
10-20 min	<ul style="list-style-type: none"> - Fill out questionnaire 	Give feedback on the workshop and the tools of the TILES toolkit (Docs section, App. Dev. Process, etc.)

Table 7-2, Non-expert workshop protocol

7.1.5 Results

7.1.5.1 Observation

Observations made during the first workshop suggested that when the participants clicked on links in the documentation section they sometimes struggled somewhat to get back to the same place in the process description. This was seen in places where the documentation refers to operations that should be carried out on the TILES Cloud web portal, and when the participants followed these links they lost the flow of the process description. In the discussion after the workshop it was suggested to open all links in the documentation section in new tabs to overcome this issue. In addition, it was observed, as well as reported in the discussion, that the transition from the TADP to the JavaScript API section and back again in step five of the process was a bit difficult to follow. Here it was suggested to put more emphasis on the navigation in the documentation, both in step six of the TADP and at the end of the JavaScript API section so that the users would be able to better follow the documentation chronologically.

Apart from these minor issues with the navigation in the documentation section, the participants of the first workshop were able to rapidly get started with the application prototyping with no help from workshop supervisors. It is important to emphasize that the participants of the first workshop were more experienced than the target non-expert user, but the fact that they could follow the flow of the process without help suggests that the process is self-supporting and enables users to go through the steps without supervision. The observations from the second workshop, where the participants were less experienced, support this conclusion, as they too were able to prototype the IoT scenario with little help from workshop supervisors.

The participants of the first workshop also declared that the TILES toolkit Documentation Section had a similar look and feel as many similar docs sections that the participants have previously used. During the first workshop, there were some connection issues with the new TILES Gateway application, but these issues were quickly overcome and did not affect the flow of the prototyping process. These issues were later resolved and did not occur in the second workshop. The code produced by the two groups of the first workshop can be seen in *Code snippet 7-1* and *Code snippet 7-2*. What we can conclude from the code snippets is that both groups were able to successfully create an application, configure their abstract TILES Squares in the web portal, and code the application in TIDE.

During the second workshop, the participants were observed to struggle some with the JavaScript syntax. This aligns with the conclusion of the specialization project [23] that the syntax of JavaScript seems to be the main issue with which non-experts are struggling with in their interaction with the TILES toolkit.

```
1 var tilesLib = require('/tiles-lib/api');
2 var client = new tilesLib.TilesClient('Petter', 'Petter_test',
   '178.62.99.218', 1883).connect();
3 var reader = new tilesLib.EventReader();
4
5 client.on('receive', function (tileId, event) {
6     /* AUTO GENERATED CODE START (do not remove) */
7     var shoe_right = reader.getTile('shoe_right', client);
8     var shoe_left = reader.getTile('shoe_left', client);
9     /* AUTO GENERATED CODE END (do not remove) */
10    var tileEvent = reader.readEvent(event, client);
11
12    if (tileEvent.isDoubleTap) {
13        shoe_right.ledOn("green");
14        shoe_left.ledOn("green");
15        if(tileEvent.name == shoe_left.name) {
16            shoe_right.hapticBurst();
17        } else {
18            shoe_left.hapticBurst();
19        }
20    }
21    if(tileEvent.isTilt) {
22        shoe_right.ledOff();
23        shoe_left.ledOff();
24    }
25 });
```

Code snippet 7-1, First workshop, application code group A

```
1 var tilesLib = require('/tiles-lib/api');
2 var client = new tilesLib.TilesClient('Ernst', 'Ernsts-Sko',
  '178.62.99.218', 1883).connect();
3 var reader = new tilesLib.EventReader();
4
5 client.on('receive', function (tileId, event) {
6   /* AUTO GENERATED CODE START (do not remove) */
7   var RighthShoe = reader.getTile('RighthShoe', client);
8   var LeftShoe = reader.getTile('LeftShoe', client);
9   /* AUTO GENERATED CODE END (do not remove) */
10  var tileEvent = reader.readEvent(event, client);
11
12  if (tileEvent.name === RighthShoe.name &&
    tileEvent.isDoubleTap) {
13    RighthShoe.ledOn('green');
14    LeftShoe.ledOn('green');
15    LeftShoe.hapticLong();
16  }
17  if (tileEvent.name === LeftShoe.name &&
    tileEvent.isDoubleTap) {
18    RighthShoe.ledOn('green');
19    LeftShoe.ledOn('green');
20    RighthShoe.hapticLong();
21  }
22  if ((tileEvent.name === LeftShoe.name || tileEvent.name ===
    LeftShoe.name) && tileEvent.isTilt) {
23    RighthShoe.ledOff('green');
24    LeftShoe.ledOff('green');
25  }
26 });
```

Code snippet 7-2, First workshop, application code group B

7.1.5.2 Survey

The answers of the survey filled out by the non-experts can be seen in this chapter. The questionnaire is divided into sections in order to systematically go through the topics of interest in the survey.

Regarding the documentation

The first section of the questionnaire was regarding the support documentation available on the docs section of the TILES Cloud web portal. As seen in the summary of the answers in *Figure 7-2* the participants were very positive to the structure and content of the documentation. We can see that almost all answers regarding the usefulness of the documentation (Q1), documentation ease of use (Q2), ease of transitioning from ideation to prototype (Q3) and usefulness of provided JavaScript examples (Q4) were in the positive end of the Likert scale. This entails that the participants found the documentation satisfactory in explaining the process and with only some difficulties. As some of the participants were very inexperienced with programming, they were struggling some with the concept of an online developers documentation section.

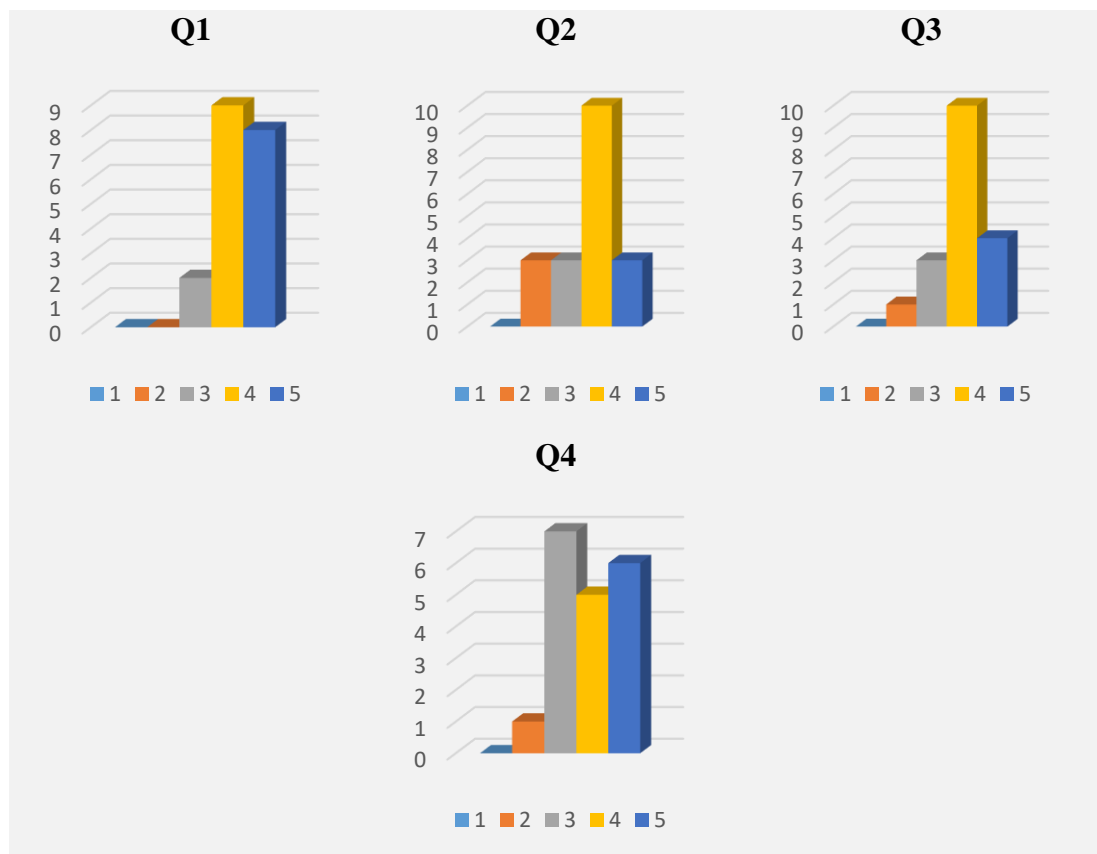


Figure 7-2, Workshop questionnaire answers, regarding the documentation

Regarding the prototyping process

The second part of the questionnaire was regarding the prototyping process specifically. Also in this section, the participants found the provided tools satisfactory as most answers was in the positive end of the Likert scale. This shows that the prototyping process (TADP) was easy to follow (Q5), there were few challenges with the process description (Q6), it was always clear what the next step was (Q8) and it was fun to apply the process (Q9). In question seven the participants were allowed to elaborate what challenges they had with the process. In the first workshop it was noted that the participants found the gateway application difficult to use, but this problem was solved for the second workshop. In the second workshop, the participants found the prototyping process slightly more difficult to follow, but based on the feedback in question seven, this problem seems to be to the fact that the participants were familiar with python and found JavaScript difficult to use.

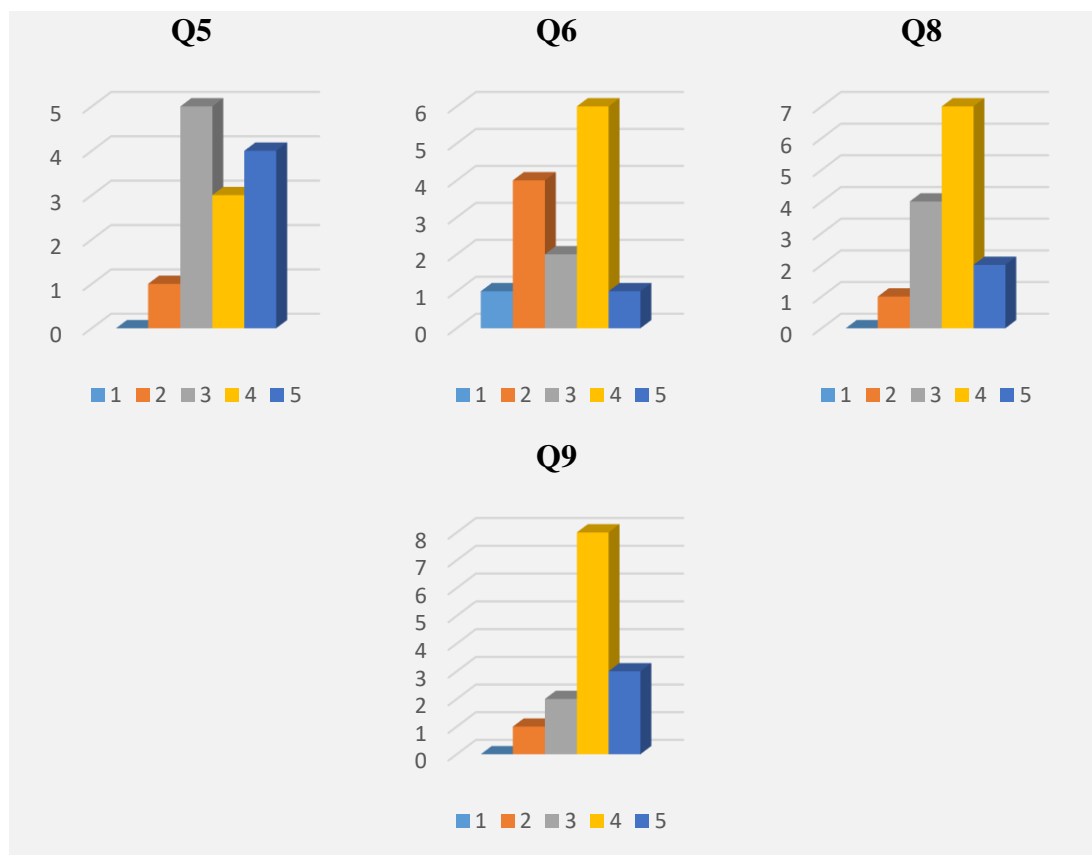


Figure 7-3, Workshop questionnaire answers, regarding the prototyping phase

Regarding the scenario

The third part of the questionnaire was regarding the provided scenario, seen in *Figure 7-1*. In the first two questions of this part, the scenario was easy to understand (Q10) and the scenario was easy to implement (Q11), there was agreement amongst the participants as almost all answers were in the positive end of the Likert scale. For the question regarding whether the participants would use this application at home if it existed (Q12), the answers were more evenly distributed as some participants did not see the need for this application.

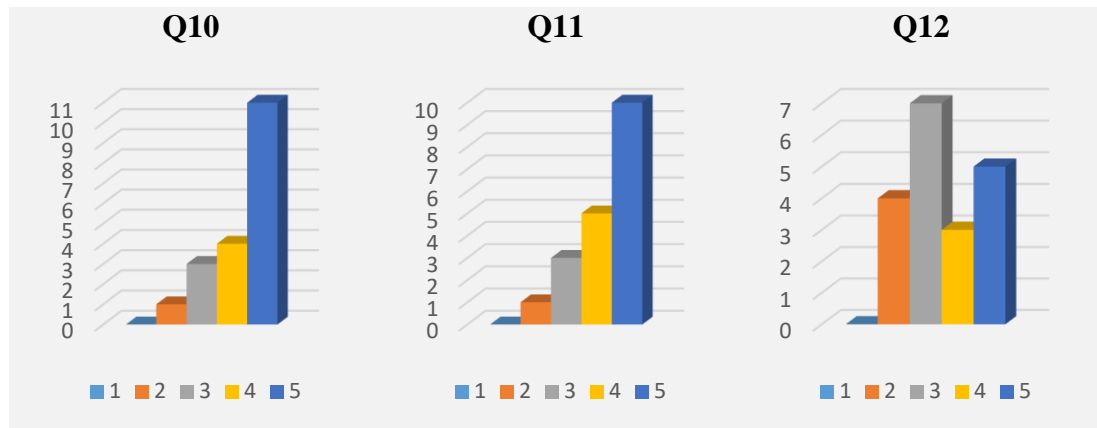


Figure 7-4, Workshop questionnaire answers, regarding the scenarios

Regarding the tools

The fourth part of the questionnaire was regarding the tools used in the workshop. Again we find most of the answers in the positive end of the Likert scale, indicating that navigating the TILES Cloud web portal was easy (Q13), using TIDE was easy (Q14) and that there were few challenges with using the TIDE (Q15). We do see, however, that some participants found the TIDE slightly difficult to use. This could be due to the fact that the participants of the second workshop were unfamiliar with JavaScript and found it difficult to manage the syntax of the programming language.

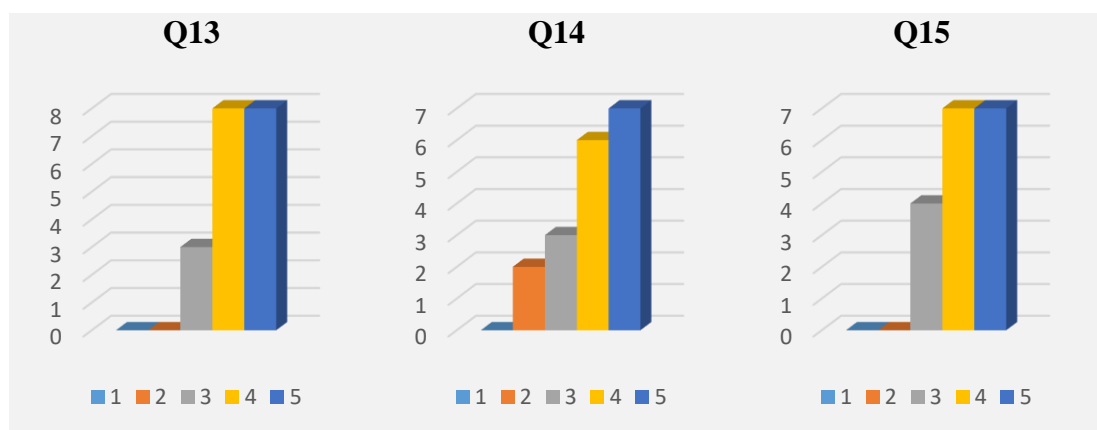


Figure 7-5, Questionnaire answers, workshop 1, regarding the tools

General

The final two questions employing the Likert scale were general questions. Q17, regarding the ease of using JavaScript in application development received very evenly distributed score on the Likert scale as seen in the figure below. Similarly, in the text questions (19-23) there were some feedback on the structure of the JavaScript API from the participants of the first workshop. Several participants noted that the naming of the template files were a bit confusing as they did not understand at first where they should write their application code. As a result, the structure of the JavaScript API template files were immediately modified based on the provided feedback such that the files would be less confusing to future workshop participants. The participants of the second workshop also struggled some with using JavaScript for development, but their problems were more related to the fact that the syntax of programming was in general unfamiliar to them.

At the end, the participants disagreed somewhat in whether they would like to have some TILES Squares at home to play with (Q18). This answer can be seen in relation with Q12, where some participants also disagreed with whether they would use the provided scenario application if it existed or not. In the future it would be interesting to see if these answers would change if more scenarios with different areas of application were available to the participants during the workshop.

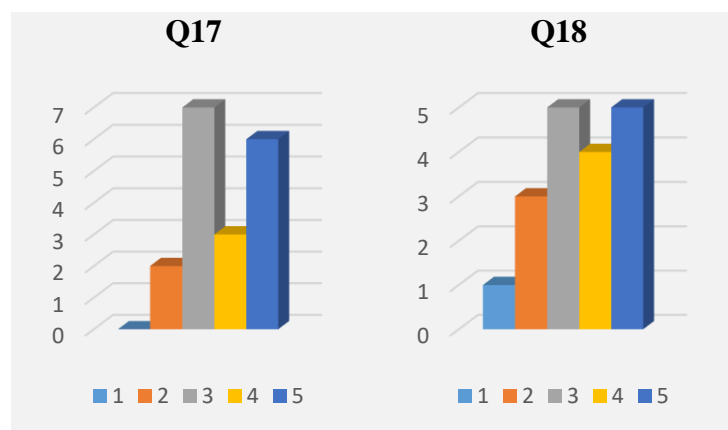


Figure 7-6, Questionnaire answers, workshop 1, general

7.2 Expert focus group

This subchapter will explain the focus group organized for evaluating the tools of the extended TILES toolkit targeting the expert users.

7.2.1 Objectives

The objective of the focus group organized with expert users was to test the TEP, TILES Cloud web portal and all the supporting documentation available to expert users in the TILES toolkit. The objective of the evaluation was not to test the experts' ability to develop firmware code for Arduino, thus sample firmware code was provided for them. The main subject for evaluation was to see whether the experts were able to extend the development APIs of the TILES toolkit in such a way that the facilitation mechanisms provided for non-experts are not broken, but made available with the new improved API capabilities implemented by the experts. Due to time constraints, the implementation provided by the expert users was not tested on non-experts, but it was evaluated to ensure that it fulfilled the required structure of the JavaScript API established in the specialization project [23].

The TEP introduced in *chapter 3* together with the *functional requirements* from *chapter 4* established the base evaluation criteria for the expert user evaluation. The hypothesis to be tested during the evaluation can be seen below. By using the documentation section available on the TILES Cloud web portal the expert users should be able to extend the TILES toolkit with minimal support from toolkit experts.

Hypothesis:

Experienced users should be able to employ TEP with its documentation in TDS to extend the TILES toolkit firmware and APIs with new improved primitives and services without breaking support for rapid application prototyping by non-experts.

7.2.2 Research tools

The research tools employed during the focus group was mainly *observation* and *discussion*. Since the focus group was conducted with a small group of expert users, it was considered that a discussion was more valuable than a questionnaire, as the expert users would be able to reflect on their experience with the TILES toolkit and compare it with similar tools they have used. The discussion was very helpful as it provided a qualitative evaluation of the TEP and TDS of the extended TILES toolkit. The expert users were also observed during the practical phase of the focus group, which provided supervisors with objective feedback on their usage of the available tools.

7.2.3 Participants

The participants of the focus group for expert evaluation were five computer science students with some programming experience, but little experience with JavaScript programming. Before the participating in the evaluation of the TILES toolkit Extension Process, they participated in a pilot workshop for non-experts, explained in *chapter 7.1*. Participating in the non-expert workshop served as an introduction into the TILES toolkit, and proved to be an excellent way for the expert users to understand how the system components were coupled together before they embarked on the tasks of extending the toolkit further by employing TEP.

7.2.4 Setting up and running the focus group

As the objective of the focus group is to evaluate the TEP, and not the participants' ability to develop and deploy firmware on the TILES Square hardware, the extended TILES Squares and corresponding hardware was implemented prior to the focus group and step one of the TEP was skipped during the focus group. The available extended hardware was the TILES Square printer that will print messages to a piece of paper, and an LED strip with more advanced display features than the LED available on the standard TILES Squares. Other than implementing the firmware on these two extended TILES Squares, no setup is required for running the focus group. The participants will be able to download all the required files from the TDS, available on the TILES Cloud server, to get started with the TEP. Since the participants had already participated in the non-expert workshop, they were able to get started with the toolkit extension without further introduction.

7.2.5 Results

7.2.5.1 Observation and discussion

In the discussion during the focus group it was reported that the participants were having some difficulties with differentiating the API clients of the JavaScript SDK. When they were asked to extend the JavaScript APIs with the new hardware features of the extended TILES Squares provided by the focus group supervisors, they did not at first realize where to write the code. This problem might simply be caused by the fact that the participants themselves had not developed the firmware of the extended TILES Squares, thus did not immediately realize that the provided services was in fact extended TILES Squares hardware and not third party services requiring additional API clients to be implemented.

Observations made during the focus group also showed that the experts were having some difficulties with the precise commands to be sent to the hardware for controlling the output primitives. Similarly to the previous problem, this problem was caused by the fact that the participants had not implemented the firmware of the extended hardware themselves, thus did not fully understand what commands were handled by

the firmware. After spending less than 10 minutes showing and explaining the firmware implementation, however, it became abundantly clear to the participants how to implement the extension methods in the JavaScript APIs and they were able to implement extension methods quickly thereafter.

7.2.5.2 Extension code

Both groups participating in the focus group decided to extend the TILES toolkit with the available LED strip. The extension code for the *EventReader* written by the two groups can be seen in *Code snippet 7-3* and *Code snippet 7-4*.

In *Code snippet 7-3*, we can see that *group A* has successfully implemented a method on the *EventReader* for abstracting the technical details of the hardware. The implemented methods enables non-experts to trigger the *rainbow* command on the LED strip, which will set the individual LEDs of the strip to different colors, as seen in *line 11* of the code snippet. The snippet also contains a method for turning the LEDs off as seen in *line 14*, which shows that the participants in *group A* understood the task, and were able to abstract both turning on and off commands to the available hardware.

```

1  EventReader.prototype.getTube = function (name, client) {
2      var id = 0;
3      if (client.tiles[name]) {
4          id = client.tiles[name];
5      }
6      var tile = {
7          name: name,
8          id: id
9      };
10
11     tile.raindbow = function() {
12         client.send(id, 'led', 'on', 'rainbow');
13     }
14     tile.ledOff = function() {
15         client.send(id, 'led', 'off');
16     }
17
18     return tile;
19 }

```

Code snippet 7-3, EventReader extension for LED strip, group A

In the second code snippet, seen in *Code snippet 7-4*, we can see that *group B* has similarly been able to abstract the details of the hardware by implementing methods in the JavaScript API for setting a random color, as seen in *line 14* in the code snippet below. In *line 11* of the code snippet we see that *group B* is trying to send a blink command to the LED strip. This however, is not a valid command for the LED strip. The fact that *group B* did not code the firmware themselves could be the reason for implementing such an erroneous method. Even with this erroneous method we can see that the structure of the extended API is implemented according to the API structure of the *EventReader*, and the code implemented by both groups was tested successfully in a simple test application.

```
1  EventReader.prototype.getLedStrip = function (name, client) {
2      var id = 0;
3      if (client.tiles[name]) {
4          id = client.tiles[name];
5      }
6      var strip = {
7          name: name,
8          id: id
9      };
10
11     strip.rainbow = function() {
12         client.send(id, 'led', 'blink', 'rainbow');
13     }
14     strip.randomColor = function() {
15         var x = Math.floor((Math.random()*3));
16         if(x ==1){
17             client.send(id, 'led', 'on', 'red');
18         }
19         if(x==2) {
20             client.send(id, 'led', 'on', 'white');
21         }
22         if(x==3) {
23             client.send(id, 'led', 'on', 'blue');
24         }
25         if(x==0) {
26             client.send(id, 'led', 'on', 'rainbow');
27         }
28     }
29
30     strip.ledOff = function() {
31         client.send(id, 'led', 'off');
32     }
33
34     return strip;
35 }
```

Code snippet 7-4, EventReader extension for LED strip, group B

7.3 Summary

In this chapter, we have seen how the workshops with non-experts and focus group with expert users have served as a baseline for evaluating the work presented in this thesis report. Based on observation and feedback from participants, the functional requirements introduced in *chapter 4* were tested, and the design and implementation elaborated in *chapter 5* and *chapter 6* was iteratively improved based on the feedback and results of the evaluation cycles. After each conducted iteration of evaluation, the results and feedback were analyzed, and modifications to the platform to improve the user experience were implemented.

For both the non-expert and expert evaluation, the hypotheses of the evaluation were put to the test and accepted as both hypotheses passed the test. Seeing these results together with the successful results of the workshops conducted for the specialization

project [23] it entails that non-experts are able to transition from the ideation phase and rapidly prototype their own IoT applications by employing the TILES toolkit systematically as described by the TADP and supported by TDS, and that expert users are able to extend the toolkit with new and improved hardware and software capabilities without breaking the support for non-expert development facilitation mechanisms provided by the TILES toolkit.

8 Related Work

In this chapter the state-of-the-art toolkits and literature related to the TILES toolkit is introduced and detailed. This chapter will first introduce available commercial toolkits evaluated during this project, before some relevant literature is introduced and elaborated. This chapter is intended to highlight the uniqueness and innovativeness of the work conducted in this research paper. The work explained in the preceding chapters of this report describes the TILES toolkit, while this chapter will look into other research and toolkits to assert where this research report and the TILES toolkit fits on the map over available IoT toolkits and research literature.

8.1 Commercial toolkits

A list of state-of-the-art IoT prototyping toolkits were presented in the specialization project [23], and can be seen in *Table 8-1*. Some of the toolkits listed in this table are, like the TILES toolkit, aiming at delivering end-to-end IoT prototyping tools with everything from cloud services, a gateway, dedicated embedded hardware devices and interfaces for programming and configuration in order to put it all together. In the table we can see that some of the toolkits are mainly intended as educational tools, while others aim at being used for prototyping and production.

Toolkit	(Prot)otype, (Prod)uct or (Edu)cational	Modularity	Programming language	End-to-end IoT
Arduino	PROT, EDU	With groove or tinker shields	Arduino IDE	yes
ARTIK	PROD	no	ARTIK IDE	yes
Bare conductive Board	PROT	no	Arduino IDE	no
Electric IMP	PROD	no	Squirrell	yes
ESP8266	PROT	no	Arduino IDE, JS	no
Gadgeteer	PROT, EDU	cable modules	.NET	no
Intel Edison	PROT	shields	Arduino IDE, C, Java, NodeJS	no
Kano	EDU	cable modules	?	partial
Lightblue Bean	PROT, PROD	no	Arduino IDE	no
Little Bits	EDU	By default	NONE	partial
Mesh	PROT, EDU	By default	Proprietary (Visual)	partial
Metawear	PROD	no	MBED	yes
Microduino	PROT	By default	Arduino IDE	no
Nordic nRF51 DK	PROD	no	C	no
Particle	PROD	extension shields	Proprietary	yes
Plezmo	EDU	By default	Scratch (Visual)	partial
Printoo	PROT	no	?	no
PuckJS	PROT	no	JS	no
Raspberry Pi	PROT, EDU	with groove or tinker shields	Phyton, JS, ...	no
ReSpeaker	PROT	shields	Arduino IDE	no
RFDuino/Simblee	PROT	extension shields	Arduino IDE	no
Sam	PROT, EDU	By default	Proprietary (Visual)	no
Tessel	PROT	with groove shield	JS	no
The air board	PROT	Shields	Arduino IDE	no

Table 8-1, Commercial toolkits for IoT prototyping [23].

Credit: Simone Mora

8.1.1 End-to-end

Arduino, ARTIK, Electric IMP, Metawear and Particle provide end-to-end IoT services in their toolkits as seen in *Table 8-1*. This means that they, similarly to the TILES toolkit, offers hardware, software, development tools and documentation for developing full-scale IoT applications without using third party services. Their level of complexity, however, is very different. As an example, Arduino is an open source

electronics platform that exist in a variety of hardware configurations. Arduino exist in the heart of countless project all over the world in addition to being the fundamental building block in the TILES Squares. Although Arduino is the only open-source end-to-end IoT toolkit platform on the list, it is too expert oriented and not suitable for non-experts. The configuration and development is cluttered with complicated procedures that cannot be expected of non-experts

The other end-to-end IoT toolkits target the production market. These toolkits are not open-source and can be costly depending on the scale of the target project. On the other side they come with customer support and facilitation mechanisms making them a lot easier for non-experts to comprehend and use. In addition, as opposed to Arduino, these four toolkits comes with support for third party web services, which makes it possible to extend and integrate the toolkits with other services. That said, however, little information exist on the toolkits ability to extend the toolkits with custom hardware, which suggest that they are poor substitutes for the TILES toolkit that is targeting toolkit extension and high degree of hardware and software customization in the toolkit.

8.1.2 Partial end-to-end

Kano, Little bits, Mesh and Plezmo are listed as partial end-to-end toolkits in *Table 8-1*. This means that the toolkits does offer their hardware components to be connected in the toolkit to work together and exchange information. However, they do not support internet connectivity for IoT and third party service integration. This means that the capabilities of the toolkits are very limited, and it is not possible to customize them to fit the general IoT scenario. In addition, they are also more oriented towards educational purposes as seen in the table, and are not suited for general application prototyping. On the other hand, if you are looking for a toolkit to use in a very specific application scenario and do not need to extend it any further, one of these toolkits might just fit your need.

8.1.3 Not end-to-end

The remaining toolkits in *Table 8-1* does not offer end-to-end nor partial end-to-end support in their services. Common for all these toolkits is that they offer their own hardware configurations and SDKs for implementing the hardware functionality. Even though these toolkits does not offer end-to-end services, they are perfect candidates to be used as building blocks or third party embedded hardware in the end-to-end toolkits. In fact, we can see that RFDuino is listed in *Table 8-1*, which is built on the Arduino structure and is the fundamental building blocks used in the TILES Squares of the TILES toolkit.

8.2 Literature

Several papers have been published on the topic of facilitating the development of IoT systems. The various research often focuses on supporting specific architectural layers, and/or specific programming approaches. This section will summarize some papers published on related topics, and a list of relevant support literature can be seen in *Table 8-2*.

In order to fit on a single page, some of the columns in *Table 8-2* have been shortened. The columns *Mod* refers to whether the system is *Modifiable* or not, while the *Wear* columns specify if the hardware configuration of the system is a *Wearable*. As we can see in the IoT-ready column, most of the presented tools are in fact not IoT-ready. Even so, they still contain interesting insight into research with non-expert users and other aspects related to this thesis paper.

Name	Year	Hardware	Language	Approach	Everyday objects	Untethered	IoT-ready	Lang. agnostic	Mod	Wear	Ref
Paper-Mache	2004	Rfid+CV	Java APIs	event	yes	yes	no	no	no	no	[12]
Calder	2004	custom	APIs	event	yes	partially	no	n/a	yes (hw)	yes?	[13]
Phidgets	2005	custom	APIs	widget	no	no	no	yes	no	no	[8]
CookieFlavors	2007	custom	Quartz (visual)	n/a	yes	yes	no	no	no	yes	[10]
Arduino	2007	custom	C/Custom	n/a	no	no	no	no	yes (hw)	yes	[16]
iStuff Mobile	2007	smartits	Quartz (visual)	event	no	yes	no	no	yes	no	[2]
VoodooIO	2007	custom	APIs	event	yes	no	no	yes	no	yes?	[29]
Siftables	2007	custom	APIs	n/a	no	yes	no	no	no	no	[17]
Blades and Tiles	2009	custom	none	n/a	no	no	no	n/a	no	no	[27]
Little Bits	2010	custom	none	n/a	no	yes	yes	n/a	yes (hw)	no	[3]
i*CATch	2010	custom	Custom (visual)	n/a	yes (wear)	yes	no	partially	no	yes	[24]
ModKit	2011	arduino	Scratch (visual)	n/a	no	no	no	no	yes (sw)	no	[18]
DUL Radio	2012	custom	Serial API	event	no	yes	no	yes	no	yes	[4]
BRIX	2012	custom	none	n/a	yes (partly)	yes	no	n/a	no	yes?	[31]
BitWear	2013	custom	Rest API	event	no	yes	yes	yes	yes (hw)	yes	[15]
Bloctopus	2015	custom	Custom (visual)	n/a	no	no	no	no	no	no	[26]
Smart its	2004	custom	Custom (visual)	n/a	no	yes	no	n/a	yes	yes	[7]

Table 8-2, Related literature for IoT and prototyping toolkits

Credit: Simone Mora

8.2.1 BitWear

BitWear is a platform for prototyping small, wireless, interactive devices. In their paper, Lyons et al. [15] are using BitWear to create, explore and experiment with wearable and deployable devices. The BitWear hardware seek to push towards the limits of small sized devices. The Murata module, built for the BitWear project, is constructed on Texas Instrument's¹⁶ CC254X chip, which measures in at an impressive 10.4x7.7x1.8mm. The Murata module is extended with a battery and an antenna for communication over BLE just like the TILES Squares. The size of the Murata module, however, enables the small battery to power the module for only a single day, making it difficult to be used in remote places where frequent access in order to replace or charge the battery can be challenging.

The Murata module features a button and an RGB LED for input and output primitives. In addition, the hardware accommodate GPIO and I2C busses for configuration of additional primitives. Since the embedded hardware is implemented to communicate over BLE, a gateway layer running on a smartphone needs to be used to convey the messages between the cloud and hardware infrastructure. Although this platform has a lot of similarities with the TILES toolkit, the BitWear project differs from the TILES project in that it does not aim at being used by non-experts. The modularity of the Murata module is limited to the abilities of expert users, and no development interface for non-experts are presented.

8.2.2 Bloctopus

Bloctopus on the other hand, is a project that enables novice users, or non-experts, to rapidly prototype tangible interfaces. The system “*enables designers to iterate design concepts, gather feedback, and learn quickly from mistakes*” [26]. In the paper, Sadler et al. addresses the issue of technical challenges for novices to create functional electronic prototypes. The authors identifies that supporting novices in application development, often comes with a tradeoff between presenting a much to simplified abstraction, such as LEGO Mindstorms [14], or require knowledge about circuit building and textual programming, such as with Arduino. The Bloctopus system is trying to position itself in between those two extremes by stating that “*the goal of our research is to better understand how we can enable novices to create electronic prototypes with improved design toolkits*” [26].

In achieving their goal, the authors interviewed experts in the area of design, prototyping, education and embedded systems, which enabled them to come up with three interesting guiding insights for developing a system targeting novices.

1. *Most Ideas are Simple Interaction*
2. *Feedback First*
3. *Making Interfacing a One-Step Process*

¹⁶ <https://www.ti.com/>

The first principles build on the discovery that “*while a design may eventually become complex in implementation, a large number of electronic prototype ideas revolve around simple interactions*” [26]. The authors state that by exposing the implementation details, the prototyping output can be significantly reduced.

The second principle detects that both novices and expert users usually start by asking the question of how the sensors of the available hardware work. This triggered the need for Sadler et al. to define a feedback loop for their system that enabled the discovery of hardware capabilities as direct and immediate as possible.

The third principle recognize the need for a standard interface for plugging in new hardware sensors to the system. “*If one has to connect many wires and add supporting circuitry before plugging in a sensor, this adds additional friction to the process of trying out new components*” [26].

The Bloctopus system, however, as seen in *Table 8-2* is not IoT-ready. This means that although the prototyping principles are important factors for consideration, the system cannot be used to design IoT application prototypes. What we can get from this paper related to the TILES toolkit is the authors’ keen insight into novice programming tools and their identification of the need to reduce the difficulty threshold for novices, and at the same time increasing the functionality ceiling in order not to get limited by the facilitation mechanisms provided for the novice users. In this regards the authors claim that there have been many attempts to solve this issue and they state that “*two types of novice programming interfaces exist: those that focus on augmenting text-based programming, and those that incorporate visual programming*” [26].

8.2.3 BRIX

BRIX [31] is a modular hardware prototyping platform for applications in mobile, wearable and stationary sensing, data streaming and feedback. Using BRIX does not require any knowledge of electronics or hardware design, and it comes with three types of stackable modules that can be connected to be used in a variety of scenarios. Zehe et al. [31] argue that novel applications often use hard-to-acquire hardware, and a timely and costly prototyping stage including several revisions. BRIX is trying to overcome these issues by allowing researchers to build and modify the hardware regardless of experience in electronics and hardware design. “*With the BRIX system we introduce a user-friendly, runtime-configurable, modular hardware prototyping platform for applications in ambient intelligence and ubiquitous computing*” [31].

During their investigations of existing solutions, the authors define six system requirements for a modular prototyping platform. The ideal system:

1. should to be *easy to use*
2. has to be *modular and easily extensible*
3. has to be *small and compact*
4. should offer a *wireless interface*
5. requires a *platform independent host software*
6. should be *low-cost and well reproducible*

Their rationale for developing the BRIX system is supported by introducing a number of state-of-the-art systems, and elaborating that although some of them support a subset of their system requirements, none of the introduced systems support all of them.

Similarly to the TILES toolkit, the BRIX system relies on Bluetooth as communication protocol between the hardware and gateway implementation. The BRIX system, however, does not support extension of the hardware capabilities, and although it allows client code to be written in any programming language supporting serial ports, no support documentation or process description exist for allowing non-experts to easily participate in the design and prototyping phase. As stated by Sadler et al. [26] there are two types of novice programming interfaces, those that augment text-based languages and those that incorporate visual programming, and BRIX seems to be doing neither. Thus the BRIX system lacks certain qualities in order to be adopted by non-experts. On the other hand, the BRIX system has successfully abstracted the hardware complexities of a modular system, thus some facilitation mechanisms do exist.

8.2.4 Functionality vs. Ease of use

A well cultivated dogma in the scientific community regarding systems targeting non-experts, and is being repeatedly mentioned in the literature listed in *Table 8-2*, is the tradeoff between functionality and power versus expressivity and flexibility [4] [18] [26]. This ideology seems to be the leading argument for research with platforms for facilitating circuit design amongst non-experts. In DUL Radio [4], i*CATCH [18], LittleBits [3] and Modkit [18] we get insight into several approaches to find the perfect balance of the two conflicting features. With DUL Radio, Brynskov et al. [4] are providing tools for creating optimized performance with regard to sketching sensor-based interaction. The platform itself consist of custom-made hardware design, as well as software, but according to the authors, their processes are applicable to other common components that can be bought in well-equipped electronic retailers.

In LittleBits [3] the author defines a vision in which designing electronic circuits is as simple as going to the store and buying a set of pre-configured components and putting them together. The aim with the platform is to move interaction with electronics from late stages of the design process to its earliest ones, in addition to enable non-experts to participate in the process.

9 Conclusions

9.1 Summary of results

This project has resulted in a comprehensive extension and improvement of the TILES toolkit. New processes for using the toolkit have been constructed together with support documentation explaining how to use them. In addition, new components have been added to the architecture and implemented programmatically, and evaluation in the form of workshops and a focus group have been conducted to evaluate and test the extended TILES toolkit and its support tools. This research has been devoted to extend the research of the specialization project [23] and aligning the research with the vision of the TILES toolkit has resulted in tools addressing the needs of the non-experts in prototyping IoT applications, in addition to creating tools aimed at facilitating expert users in their need to extend and customize the toolkit for specific IoT scenarios. The following subsections will discuss the research questions of this research paper and summarize the results and contributions of this research project.

9.1.1 RQ1: How to support rapid prototyping and deployment of IoT applications using TILES?

This research question is derived directly from the vision of the TILES project regarding rapid prototyping of IoT applications based on everyday object augmentation. It is the main research question on which this research project is built. To answer this broad question, it was divided into three sub-questions that were answered through formalizing process description, requirement analysis, design science and user studies. New processes were formalized for the two main users of the TILES toolkit for supporting the usage of the tools and enabling rapid prototyping and toolkit extension.

9.1.2 RQ1.1: How to support rapid prototyping of ideas created with TILES Cards?

In order to support rapid transition from ideation to prototyping, support tools had to be formalized and added to the TILES toolkit. Before this project, there was an

inconsistency in the process of prototyping an application based on ideas generated during the ideation process. Transitioning from the idea to the program code of the application required supervision from toolkit experts, and non-experts were not able to transform the ideas into program code on their own. By introducing the self-supporting TADP in the toolkit with detailed instructions on how to employ the process in the TDS, non-experts were able to rapidly transition from ideation to prototype without toolkit experts as proven in the workshop explained in the evaluation chapter. Throughout the step-by-step guide of the TADP in the TDS, strong emphasis has been put on the transition from the TILES Cards to the application program code.

9.1.3 RQ1.2: How to support TILES application development by non-experts?

Together with RQ1.1, the TADP and TDS were introduced to address this research question. Following the TADP, with the step-by-step process description available in the TDS, the non-experts are able to develop IoT applications using the TILES toolkit without help from toolkit experts. Introducing the TIDE into the application development phase of the TADP was also an important aspect in answering this research question. TIDE enabled non-experts to rapidly get started with application code implementation, without having to rely on third party software, or downloading the whole JavaScript API to their development machines. TIDE was able to successfully abstract the complex behavior of setting up the development environment with references to the proper modules in the JavaScript API. In addition, by setting up TIDE to run on the TILES Cloud server, the application code is already stored on the server, enabling the application to be tested and deployed by the non-experts, without involving toolkit experts in the process. These facilitation mechanisms enables non-experts to use the TILES toolkit in everything from the ideation phase, prototyping phase and even to test, debug and deploy the applications.

9.1.4 RQ1.2: How to support toolkit extension for expert users with minimal efforts without breaking development support for non-experts?

In order to align this research with the vision of the TILES toolkit, to enable the tools to be used in a wide variety of IoT application scenarios and provide a system with a high degree of customization, the expert users must also be considered. In order to answer this research question, the second process (TEP) detailed in this research paper was introduced into the TILES toolkit. The process description, available in the TDS, enables expert users to follow a carefully constructed process in order to extend and customize the TILES toolkit with additional hardware capabilities in the TILES Squares, and software capabilities in the available development APIs. The structure of the extended TILES toolkit has been designed with both the non-experts and expert users in mind in order to create a viable future for the toolkit, and making it able to adapt to the ever changing demands of the rapidly changing domain of IoT.

9.2 Discussion

The processes (TADP and TEP) defined early in this paper was constructed through literature review, requirement analysis and analysis of findings of the specialization project [23]. These processes serves as an excellent baseline for the extended TILES toolkit. By deriving the TADP and TEP from the RapIoT process [21], the well-defined properties of the RapIoT process is inherited by the TILES toolkit. The work and research explained in this paper has transformed the TILES toolkit from a research oriented tool, into a semi self-supporting platform, enabling both non-experts and expert users to exploit its services to develop fully functional application prototypes and construct extended IoT prototyping toolkits for specific application scenarios with minimal support from toolkit experts.

The user studies organized as a part of this thesis has provided some interesting insights into how the TILES toolkit can facilitate IoT application prototyping for non-experts, as well as into what tools expert users need for creating customized IoT toolkits for specific application domains. On one hand, from an academic perspective, the user study contributes to the literature on what considerations are necessary when implementing a toolkit for non-expert IoT prototyping. The findings demonstrate the delicate balance between facilitation tools for non-experts and advanced tools for expert users. The needs for non-experts and expert users often do not coincide as facilitation mechanisms usually comes at a certain cost in terms of the functionality expected by the advanced user. In addition, the TILES toolkit Application Development Process developed in this project, provides an interface for further research of IoT application prototyping by non-experts. The findings of the study verified that participants were able to rapidly prototype IoT applications by following the carefully constructed development process, without supervision from toolkit experts.

On the other hand, considering the fact that the participants of the user study were able to transition from an IoT application idea, generated with the TILES Cards, into a fully functional application prototype, suggests that the TILES toolkit is ready to leave the comfortable realm of pure academic research and be applied in more practical usage areas. With a functioning toolkit prototype, seen in relation with the fact that user studies have verified the toolkit's ability to be extended and customized by experts and being used by non-experts in prototyping, it is conceivable to imagine the TILES toolkit being employed in a wide variety of both educational and production scenarios in the future.

At the same time, there are also some limitations with the findings of this project. Firstly, due to time constrains, the extended TILES toolkit with processes and support documentation was never tested as a whole on non-expert users. This implies that the findings does not adequately represent the whole target usage area of the toolkit, even though all components of the toolkit have been tested separately during this thesis project and the specialization project together. A number of authors have argued for the need for studies *in-the-wild*, especially in the field of ubiquitous computing [5],

which could provide the necessary evaluation plot for the whole extended TILES toolkit. In this setting, the expert user evaluation could even be evaluated by non-experts, obliging expert users and non-experts to evaluate each other's work, thus putting the whole system to the test. Such a case study would be extensive and time consuming, but would provide detailed reflection on the usage opportunities and capabilities of the TILES toolkit.

At last, it is important to acknowledge that numerous answers to the research questions of this project exist. An initial attempt to extend the TILES toolkit with additional tools for supporting TILES application prototyping by non-experts were made by creating the Rule Engine Development Environment. This tool was intended to be evaluated with the research question about supporting TILES application development by non-experts, but due to time constraints, this evaluation was not finalized. Extending this non-textual development environment further could be an alternative facilitation mechanism to the JavaScript development API currently supported by the TILES toolkit. The Rule Engine does already enable non-experts to define rules for their program rather than write the program instructions in JavaScript. However, as the Rule Engine has not been properly evaluated, it is difficult to see how the non-experts will accept this tool and how it can be used for their application prototyping. This could definitely be an aspect to consider in future research and usage with the TILES toolkit. Although the Rule Engine Development Environment has not been properly evaluated by non-experts, it does provide important insights into how the TILES toolkit can be extended with new development environments without conflicts for the already existing tools. This entails that the toolkit might support numerous development environments in the future, such as text based with JavaScript, rule based, visual programming, etc.

9.3 Future work

In this paper, an extensive extension of the TILES toolkit has been finalized, and the toolkit has transitioned from a research oriented tool into a self-supporting platform for IoT application prototyping. At this point in time there are several interesting pathways to take for the future work with the TILES toolkit. Most interestingly, future work could focus on continue on the line of academic research by organizing user studies *in-the-wild* to evaluate the toolkit in full-scale scenario use cases. Another approach is to leave the realm of pure academic research and take the step towards introducing the TILES toolkit into real practical usage areas and evaluate its usefulness in real IoT scenarios. This section will highlight what future work opportunities are most relevant to pursue based on the findings of this paper.

9.3.1 Bring TILES to the classroom

Until now, several workshops involving high-school students as non-experts have been conducted. These workshops have yielded promising results in the field, enabling

young students with little programming experience to successfully prototype their own IoT applications. Taking these results further, an interesting opportunity for future research is to bring the TILES toolkit into the classrooms and do a more systematic evaluation of the learning outcome of using the TILES toolkit.

9.3.2 Bring TILES to the market

Similar to the previous subsection, this future work opportunity is in realizing the market value in the TILES toolkit by bringing the toolkit out of the educational and academic sphere, and evaluate the needs of the market in order to provide a toolkit ready to be used in real life scenarios all over the world. This would entail spending a significant amount of efforts on organizing an initial transition into the production market by cooperating with commercial businesses and evaluate their needs and usage of the IoT prototyping toolkit.

9.3.3 Study *In-the-wild*

A very interesting opportunity for future work is to take the current version of the TILES toolkit, and organize studies in-the-wild, as discussed in the *Discussion* subchapter. This would provide important feedback on how the components of the TILES toolkit fit together in a real or semi-real IoT scenario, where expert users will tailor the toolkit to a certain application domain, before non-experts will prototype applications for the specific application usage areas.

9.3.4 Toolkit improvements

The three preceding future work opportunities focus on evaluation of the TILES toolkit in three various evaluation settings. The findings of this paper has also provided a list of possible toolkit improvements that will add new improved tools to the TILES toolkit.

9.3.4.1 Improving TILES Gateway

Even though the new TILES Gateway has an improved user interface and comes with some new features, support for application appropriation amongst others, the application does not seem to work properly on all version of Android. In addition, sometimes when pairing physical and virtual TILES in an application, it takes some time before the app manages to start sending and receiving events. Fixing these issues in addition to implementing the Gateway to run on additional platforms like the Raspberry Pi would be an important improvement for the TILES toolkit.

9.3.4.2 Using gateway as input/output primitive

Since all communication between the TILES Squares and the TILES Cloud interface is already pass through the Gateway layer, a possibility is to implement output and input primitives directly on the Gateway layer. This would enable the TILES toolkit to use all hardware capabilities of the smartphone, such as GPS, gyroscope, speakers and the screen. By introducing these capabilities as data primitives in the toolkit, new and exciting research possibilities with the TILES toolkit would be possible.

9.3.4.3 Shared TILES Squares

In the current implementation, a TILES Square can only be used in one application at a time. This means that if a TILES Square with an air quality sensor is deployed somewhere, this could only be used to send data to one specific application. One promising future extension of the TILES toolkit is to enable shared TILES Squares. The structure envisioned contains four types of Squares, elaborated below.

1. Private Square in One Application (PSOA)
2. Private Square in Multiple Applications (PSMA)
3. Public Square with Private Output (PSAO)
4. Public Square with Shared Output (PSSO)

The PSOA variant listed above, is what the TILES toolkit supports today. This enables a user to pair a TILES Square with an abstract Square in an application, which will lock that physical TILES Square to the specific application until the application releases the TILES Square.

The PSMA solution would enable a user to configure a TILES Square that would be shared amongst his private application. This is very similar to the PSOA, as it does not enable other users to share the data it produces, but the capabilities of the TILES Squares can be shared amongst multiple private applications. This Square type could for example be used if a user only has access to one Square with printing capabilities, but needs multiple applications to be able to print.

In PSAO it will be possible to configure a TILES Square to be publicly accessible, but with private output capabilities. This means that anyone can read the input primitives such as tap and tilt, but only the Square owner would be able to control the output primitives.

Lastly, in PSSO, all input and output capabilities are publicly available such that anyone can read the input primitives as well as write to the output primitives of the TILES Square.

9.3.4.4 Implement security

In the current version of the TILES toolkit, there is no implemented security. This means that everything from the TILES Cloud web portal, to the events and commands

of running TILES applications are publicly available to everyone. This also entails that if someone knows your *username*, *application id* and *device id*, they will be able to both read and write to the data primitives of your TILES Square. Securing the various components of the TILES toolkit was part of the requirement specification, but due to time constraint and the time consuming nature of implementing and testing security features, these requirements were discarded. Even so, however, all the components of the toolkit are designed to support future security implementation. This entails that no functionality or background services needs to be modified in order to introduce modern security mechanisms into the TILES toolkit. The MQTT and HTTPS protocol used for conveying messages between the TILES Cloud and the other layers of the toolkit, even the TILES Cloud web portal, are able to be properly secured.

9.3.4.5 Third party hardware

Adding support for third party hardware is another feature that would spawn future research opportunities with the TILES toolkit. As detailed in *chapter 8*, many toolkits providing custom hardware exist, and adding the ability to exploit these hardware capabilities in the TILES toolkit would open up to many exiting new data primitives without having to extend the hardware and firmware of the TILES Squares. The important initial step towards supporting third party hardware has already been taken by adding abstract placeholders for the TILES Squares. These abstract TILES Squares could be modified to be placeholders to any kind of third party embedded hardware.

9.3.4.6 Testing and extend Rule Engine

The extensiveness of this project and time constraints has resulted in not being able to arrange a separate user study for the Rule Engine development environment. A future research opportunity is thus to test the environment to determine how non-experts would accept it as an alternative to the JavaScript development that has already been evaluated by non-experts. In this direction of research, it would also be possible to implement an automatic transition from the Rule Engine to the JavaScript development environment, which would enable non-expert users to define all their rules in the Rule Engine, before exporting it as a JavaScript template, and continue the development with TIDE. This would provide interesting possibilities in academic research, by evaluating how the non-experts can learn to program with JavaScript by transitioning from the Rule Engine to the TIDE development environment.

References

- [1] Aursand, A., Gran, M., Johansen, A., Midboe, K., Nornes, A. & Sandell, E. (2017). IT2901 – Informatics Project II, TILES IDI – Mobile Gateway. *Internal Report at NTNU*. Trondheim, Norway.
- [2] Ballagas, R., Memon, F., Reiners, R. & Borchers, J. (2007). iStuff Mobile: Rapidly Prototyping New Mobile Phone Interfaces for Ubiquitous Computing. *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'07)*. San Jose, California, USA. Pp. 1107-1116.
- [3] Bdeir, A. (2009). Electronics as material: littleBits. *In Proceedings of the 3rd International Conference on Tangible and Embedded Interaction (TEI'09)*. Cambridge, United Kingdom. Pp. 397-400.
- [4] Brynskov, M., Lunding, R. & Vestergaard, L. S. (2012). The Design of Tools for Sketching Sensor-Based Interaction. *In Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction (TEI'12)*. Kingston, Ontario, Canada. Pp 213-216.
- [5] Bødker, S. (2015). Third-wave HCI, 10 years later—participation and sharing. *Interactions*, 22(5). Pp. 24-31.
- [6] Davidson, R., Akiba, C., & Townsend, K. (2014). Getting started with Bluetooth low energy: Tools and techniques for low-power networking. Bloomington, IN, United States: O'Reilly Media.
- [7] Gellersen, H., Koruem, G. & Schmidt, A. (2004). Physical Prototyping with Smart-Its. *IEEE Pervasive Computing*, 3(3). Pp. 74-82.
- [8] Greenberg, S. (2004). Collaborative Physical User Interfaces. *Report from University of Calgary*. Calgary, Alberta, Canada.

- [9] Hevner, A., & Chatterjee, S. (2010). Design Science Research in Information Systems. In *Design Research in Information Systems*, vol. 22. Boston, MA: Springer US. Pp. 9-22.
- [10] Kimura, H., Okuda, Y. & Nakajima, T. (2007). CookieFlavors: Rapid Composition Framework for Tangible Media. In *Proceedings of Next Generation Mobile Applications, Services and Technologies (NGMAST'07)*. Cardiff, UK.
- [11] Kirkemyr, J. (2016). End User Programming for TILES: Methods and Tools. *Master thesis at NTNU*. Trondheim, Norway.
- [12] Klemmer, S., Li, J., Lin, J. & Landay, J. (2004). Papier-Mâché: Toolkit Support for Tangible Input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'04)*. Vienna, Austria. Pp. 399-406.
- [13] Lee, J., Avrahami, D., Hudson, S., Forlizzi, J., Dietz, P. & Leigh, D. (2004). The calendar toolkit: wired and wireless components for rapidly prototyping interactive devices. In *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques (DIS'04)*. Cambridge, MA, USA. Pp. 167-175.
- [14] Lego Mindstorms, LEGO ® (2014). (www.lego.com).
- [15] Lyons, K., Nguyen, D., Seki, S., White, S., Ashbrook, D. & Profita, H. (2013). BitWear: A Platform for Small, Connected, Interactive Devices. In *Proceedings of the adjunct publication of the 26th annual ACM symposium on User interfaces software and technology (UIST'13)*. St. Andrews, Scotland, United Kingdom. Pp 73-74.
- [16] Mellis, D. A., Banzi, M., Cuartielles, D., & Igoe, T. (2007). Arduino: An Open Electronics Prototyping Platform. In *Proceedings of the Conference on Human Factors in Computing (CHI'07)*. San Jose, California, USA.
- [17] Merrill, D., Kalanithi, J. & Maes Pattie. (2007). Siftables: Towards Sensor Network User Interfaces. In *Proceedings of the 1st international conference on Tangible and embedded interaction (TEI'07)*. Baton Rouge, Louisiana. Pp. 75-78.
- [18] Millner, A. & Baafi, E. (2011). Modkit: blending and extending approachable platforms for creating computer programs and interactive objects. In

- proceedings of the 10th International Conference on Interaction Design and Children (IDC'10)*. Ann Arbor, Michigan, USA. Pp. 250-253.
- [19] Mora, S., Asheim, J., Kjøllesdal, A., & Divitini, M. (2016). Tiles Cards: a Card-based Design Game for Smart Objects Ecosystems. *Workshop on Smart Ecosystems cReation by Visual dEsign. CEUR Proceedings*.
- [20] Mora, S., Divitini, M., & Gianni, F. (2016). TILES: an Inventor Toolkit for Interactive Objects. *In Proceeding of the International Conference on Advanced Visual Interfaces (AVI)*. ACM.
- [21] Mora, S., Gianni, F., & Divitini, M. (2016). RapIoT Toolkit: Rapid Prototyping of Collaborative Internet of Things Applications. *In proceedings of the International Conference on Collaboration Technologies and Systems (CTS)*.
- [22] Mora, S., Gianni, F., & Divitini, M. (2017). Tiles: A Card-based Ideation Toolkit for the Internet of Things. *Proceedings of DIS conference*.
- [23] Mæhlum, A. R. (2016). TILES Toolkit: usage and ideation workshops and integration in IoT ecologies. *Specialization project report at NTNU*. Trondheim, Norway.
- [24] Ngai, G., Chan, S., Ng, V., Cheung, F., Choy, S., Lau, W. & Tse, J. (2010). i*CATch: a scalable plug-n-play wearable computing framework for novices and children. *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'10)*. Atlanta, Georgia, USA. Pp. 443-452.
- [25] Oates, B. J. (2005). *Researching information systems and computing*. London: Sage publications.
- [26] Sadler, J., Durfee, K., Shluzas, L. & Bilkstein, P. (2015). Bloctopus: A Novice Modular Sensor System for Playful Prototyping. *In Proceedings of the 9th International Conference on Tangible, Embedded, and Embodied Interaction (TEI'15)*. ACM, New York, NY, USA. Pp. 347-354.
- [27] Sankman, R., Ullmer, B., Ramanujam, J., Kallakuri, K., Jandhyala, S., Toole, C. & Laan, C. (2009). Decoupling Interaction Hardware Design Using Libraries of Reusable Electronics. *In Proceedings of the 3rd International Conference on Tangible and Embedded Interaction (TEI'09)*. Cambridge, UK. Pp. 331-337.

- [28] Sivapalan, V., & Kirkemyr, J. (2015). Event-driven infrastructure for the Internet of Things supporting rapid development. *Specialization project report at NTNU*. Trondheim, Norway.
- [29] Villar, N. & Gellersen, H. (2007). A Malleable Control Structure for Softwired User Interfaces. *In Proceedings of the 1st International Conference on Tangible and Embedded Interaction (TEI'07)*. Baton Rouge, Louisiana. Pp. 49-56.
- [30] Wessel, M. (2016). The Tiles Workshop: Internet of Things Workshop for Teenagers. *Specialization project report at NTNU*. Trondheim, Norway.
- [31] Zehe, S., Grosshauser, T. & Hermann, T. (2012). BRIX - An Easy-to-Use Modular Sensor and Actuator Prototyping Toolkit. *4th International Workshop on Sensor Networks and Ambient Intelligence*. Lugano, Switzerland. Pp. 817-822

Appendix A

A Deploying TILES Cloud with Cloud9 as TIDE

The first part of this appendix, regarding setting up DigitalOcean to host the TILES Cloud, is copied from the specialization project for completion. The second part, regarding the Cloud9 IDE as TIDE, is new in this paper.

A.1 About DigitalOcean

DigitalOcean is a cloud infrastructure provider, providing Infrastructure as a Service (IaaS), that focuses on simplifying web infrastructure for software developers. In this project, *DigitalOcean* was used for hosting the TILES Cloud infrastructure on a virtual machine running Ubuntu 16.

A.2 Setting up droplets in DigitalOcean

Sign in to the *DigitalOcean* web portal and navigate to the *Create Droplet* interface. The TILES Cloud droplet was set up with the configuration seen in *Figure A-1*. No *additional options* were chosen, but *adding an SSH key* is advised for additional security. The *chosen hostname* was set to *andersriTiles*, and the *Create* button was pressed.

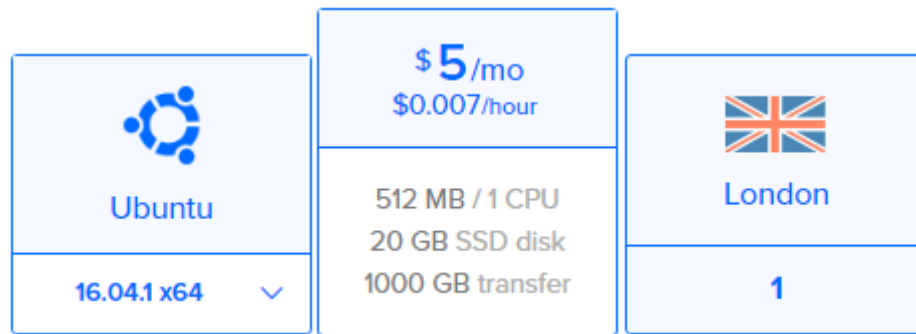


Figure A-1, DigitalOcean droplet configuration

Next step was to log into the web console, or using an SSH client to connect to the Ubuntu VM running in *DigitalOcean* and go through the steps in the following subsections.

A.2.1 Install Node.js and npm on Ubuntu via terminal

To install Node.js and npm on Ubuntu using the terminal, the following commands were fired in the terminal sequentially.

```
1 sudo apt-get update
2 sudo apt-get install nodejs
3 sudo apt-get install npm
```

Code snippet A-1, Installing Node.js and npm in Ubuntu VM

A.2.2 Cloning TILES Cloud git repository

The next step was to clone the git repository of the TILES Cloud. This was achieved by running the commands seen in *Code snippet A-2*, in the terminal. The first line will install git if it is not already installed. The second line will clone the git repository. The third line is used to navigate into the cloned repository, while the fourth line will install all required npm modules for the TILES Cloud.

```
1 sudo apt-get install git
2 git clone https://github.com/simonem/tiles-dev-sw
3 cd tiles-dev-sw/Tiles\ CLOUD/api-client
4 npm install
```

Code snippet A-2, Cloning git TILES repository in Ubuntu VM

A.2.3 Installing MongoDB

MongoDB is an open source, document-oriented, database. MongoDB is great when working with JavaScript, as it can store JSON objects in a document-oriented manner, thus MongoDB is the preferred backend database for the TILES Cloud infrastructure. The steps for installing MongoDB in the Ubuntu VM, is seen below.

```
1 sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
2 echo "deb http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
3 sudo apt-get update
4 sudo apt-get install -y mongodb-org
```

Code snippet A-3, Installing MongoDB in Ubuntu VM

In addition, the configuration file *mongodb.service* must be edited. Type the command seen below.

```
1 sudo nano /etc/systemd/system/mongodb.service
```

Code snippet A-4, Edit mongodb.service in Ubuntu VM

In the opened file editor, paste in the text in the following snippet.

```
[Unit]
Description=High-performance, schema-free document-oriented database
After=network.target

[Service]
User=mongodb
ExecStart=/usr/bin/mongod -quiet -config /etc/mongod.conf

[Install]
WantedBy=multi-user.target
```

Code snippet A-5, mongodb.service configuration file in Ubuntu VM

Now we need to start the service with `systemctl` then make sure that MongoDB will be restarted when the system starts by enabling the service. This is achieved with the following commands.

```
1 sudo systemctl start mongodb
2 sudo systemctl enable mongodb
```

Code snippet A-6, Starting MongoDB as a service

A.2.4 Running Node.js server as a service using PM2

To run the Node.js server code as a service we will use the production process manager PM2 for Node.js. First, we need to install PM2 globally, and then start the node server with PM2. In the code snippet below, we first install PM2 globally in the first line, before we navigate into the *bin* folder of the TILES Cloud repository that we cloned from git earlier. By typing the command seen in the third line, we start the www Node.js server with PM2. The last command sent is to make sure that the pm2 service is started with the system. This is all we need to do to let the system handle the Node.js server.

```
1 sudo npm install -g pm2
2 cd bin
3 pm2 start www
4 pm2 startup systemd
```

Code snippet A-7, Starting server with PM2

A.3 Installing Cloud9 core

To install the cloud9 core, some other modules must be installed by firing the following commands sequentially in the CLI of the server. The following lines will amongst other install python. In addition, *forever* will be used to host the individual TIDE clients as services, thus this is installed as seen in *line 3*.

```

1 sudo apt-get install build-essential
2 sudo apt-get install apache2
3 npm install forever -g
4
5 wget https://www.python.org/ftp/python/2.7.12/Python-2.7.12.tgz
6 tar -xvf Python-2.7.12.tgz
7 cd Python-2.7.12
8 ./configure
9 make
10 make install

```

Code snippet A-8, Installing prerequisites for Cloud9 core

The next step of the installation process of Cloud9 core is to clone the git repository and run the install scripts.

```

1 git clone git://github.com/c9/core.git c9sdk
2 cd c9sdk
3 scripts/install-sdk.sh

```

Code snippet A-9, Cloning Cloud9 core and install

Finally, we have to edit the apache2 configuration file so that the TIDE clients can be hosted.

```

sudo nano /etc/apache2/sites-available/dev.conf

<VirtualHost *:80>
    ServerName dev.yourdomain.org
    ProxyPass / http://0.0.0.0:8181/
    ProxyPassReverse / http://0.0.0.0:8181/
    ProxyPreserveHost On
</VirtualHost>

sudo a2ensite dev

```

Code snippet A-10, apache2 configuration for Cloud9

Now we are ready to start an instance of the Cloud9 environment. Navigate into the c9sdk folder containing the Cloud9 core repository and type the following command in the CLI.

```

forever start server.js -p 8181 -w ~/workspace/ -l 0.0.0.0 --auth
username:passswd

```

Code snippet A-11, Starting an instance of Cloud9 server (TIDE)

This command will start a new instance of the Cloud9 server at port *8181* with the *workspace* directory as local repository. The *--auth* keyword can be used to specify a username and password for the environment. In the TILES Cloud server, an instance

of the TIDE should not be started with *root* user as that will enable the developers to have access to root privileges. A new user *c9sdk* was created on the TILES Cloud server to start all instances of the TIDE.

Appendix B

B TILES toolkit Documentation Section

B.1 Getting Started

Introduction to the documentation

This documentation section will show you how to use the TILES toolkit to make interactive objects for learning and play. It will guide you through everything from the moment you acquire your first TILES hardware device, to the moment you are ready to launch your IoT application and ship it as a product. For more information, please visit the TILES toolkit homepage

This Guide is broken down into five parts:

- **Getting Started** goes over the basics of what you need for development and how the TILES toolkit works. If you are not an expert, and you have never played with embedded hardware devices or programmed your own application, this is the section for you. If you are an expert user and eager to get started with development, you can skip this section to get your hands dirty right away. You will always be able to revisit this section later.
- **Application Development Process** explains the development process that will guide you as a developer in turning your application ideas into working prototypes ready to be used by end users. Whether you are new to programming and want to create your first IoT application, or you are an experienced programmer that want to use the TILES toolkit in your latest project, this is the section for you.
- **JavaScript API supplements** the 5. Code Application phase of the Application Development Process. This is intended for you if you choose to use JavaScript for developing your TILES application. This section also contains code samples for JavaScript.

- **TILES toolkit Extension Process** details the process of extending the TILES toolkit with new features to be used in your own projects. This section is intended for the experienced user that wants to extend the TILES toolkit with new features such as new hardware capabilities for the TILES Squares, or new API features for the development environments.
- **Rule Engine API supplements** the 5. Code Application phase of the Application Development Process. This is intended for you if you choose to use Rule Engine for developing your TILES application. This section also contains samples for Rule Engine applications.

The best way to use this guide depends on your goal. If you are a non-expert intending to develop your first IoT application, you should:

1. Go through Getting Started
2. Go through the Application Development Process
3. Use JavaScript API for development

If you are an expert user intending to extend the toolkit you should:

1. Get familiar with Getting Started
2. Go through the TILES toolkit Extension Process

Getting Started

What do you need?



Figure B-1, What you need for IoT prototyping with the TILES toolkit

- TILES Square(s)
- TILES Gateway
- Development PC

To start developing with the TILES toolkit you will first need to procure TILES Squares and install the Gateway app on a smartphone device. Please visit the TILES toolkit homepage for instructions on how to procure TILES Squares and the TILES Gateway app. In addition you will need a development PC with an internet connection for configuring the application in the TILES Cloud web portal, and coding the

application logic. The TILES toolkit is platform independent, which means that it is compatible with any operating system you would want to use.

If you want to extend the TILES Square firmware, you will also need to download the Arduino IDE.

TILES toolkit

Architecture

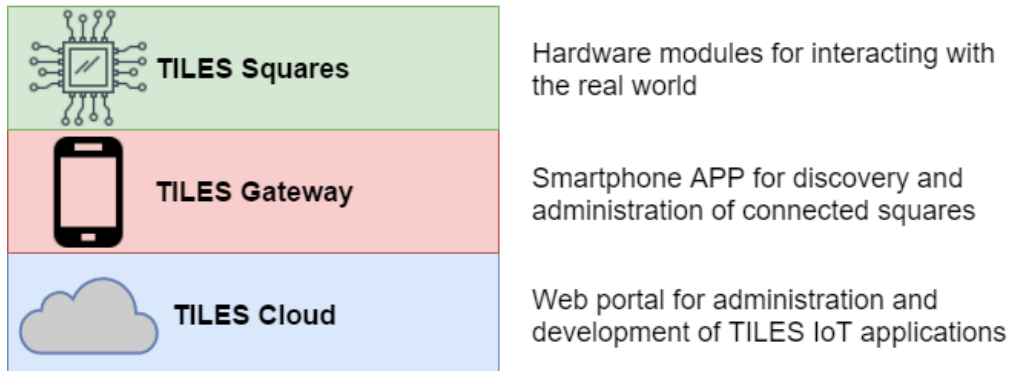


Figure B-2, Layered architecture of the TILES toolkit

The architecture of the TILES toolkit is divided into three layers as seen on the figure above. Below is a short description of the responsibilities of each layer of the TILES toolkit.

TILES Squares

The TILES Squares are the embedded hardware devices of the TILES toolkit, implemented as the immediate layer for end user interaction. Users of any TILES application will tap, tilt and otherwise interact with the TILES Squares to produce events that will be sent to the TILES Gateway, which will forward the events to the TILES Cloud server for processing. TILES Squares features several data primitives for input and output interaction with the TILES toolkit.

TILES Gateway

The TILES Gateway is responsible for forwarding the events between the TILES Squares and the TILES Cloud. The Gateway will communicate with the squares through Bluetooth, while the communication with the server is performed over the Internet. The TILES Gateway runs on any regular internet enabled smartphone with Bluetooth support. This makes it easy to carry around as you interact with your TILES Squares.

TILES Cloud

The TILES Cloud is the brains behind the TILES toolkit. This is where the application logic will run, and where the program code is executed. The TILES Cloud will receive TILES Square input events from the connected Gateways, and will process the events and reply to the Gateway with output commands that will set the output primitives of

the TILES Squares. The TILES Cloud also features interfaces for connecting your TILES application to available web services, which enables your TILES application to be integrated into your everyday life.

Development Environments

This section of the documentation is intended to help you decide which development environment you should use to develop your application. It will briefly explain the differences between the available environments and what prerequisites are needed for using them. This section will not go into details about how to develop TILES applications using the APIs of the development environments. For that purpose, see the documentation on the respective API of the development environment in question. If this is your first time developing a TILES application, we suggest the Cloud Development Environment.

Cloud development environment

The Cloud development environment is running a cloud9 web portal as your IDE. When the Cloud development environment is selected for developing a TILES application, the TILES API library files are created and the IDE is configured automatically. The cloud9 IDE will be running in the browser, which means that no local setup is required, and you can run the development environment using your favorite operating system and web browser. Once the environment is created, you will be developing your application using JavaScript, and the TILES toolkit JavaScript APIs.

To start the development environment, navigate to the list of applications and select your application. In the application details page you will need to click the Host application button, which will start a new instance of the cloud9 web IDE and display the Launch Environment button as seen in the image below.

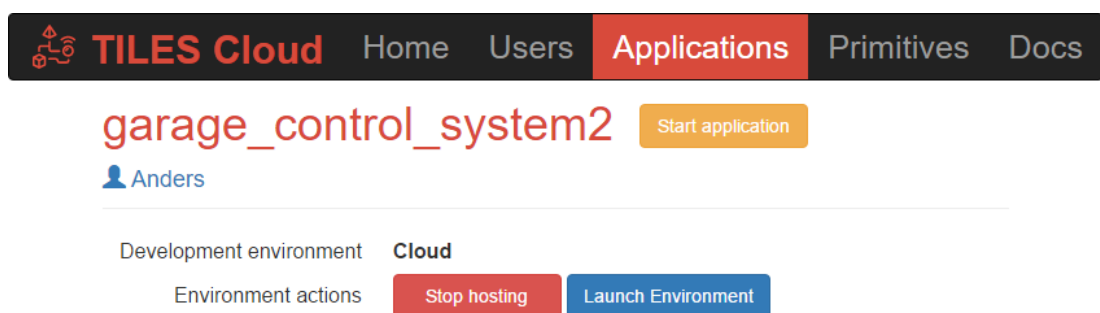


Figure B-3, Start hosting of the cloud9 web IDE for the application

Once the workspace environment has been hosted, you can access the IDE by clicking on the Launch Environment button.

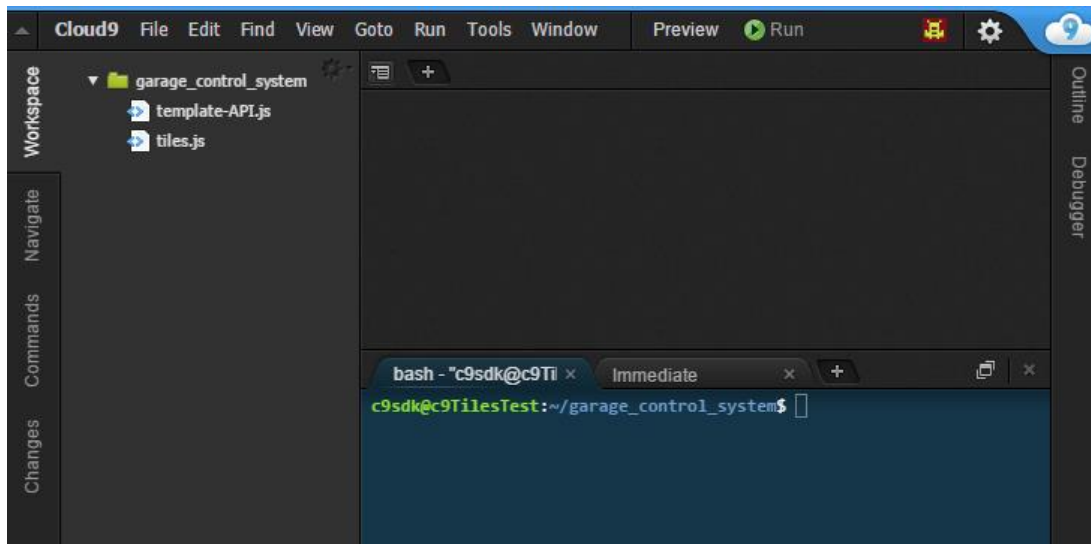


Figure B-4, Cloud9 web IDE configured with TILES JavaScript API

In the cloud workspace you will find two files:

1. **example-API.js** - Shows code examples and API instructions
2. **tiles.js** - Empty file where you will write your own application code

In addition to programming the application, you will be able to select your code file, and hit 'Run' in the IDE. This will start the program code in debug mode, and you will be able to see the output of your program in the bash window. This way you are able to properly debug your application code directly in the Cloud IDE during development. For more detailed instructions, please read through the section for Application Development Process.

Local development environment

When you select the Local development environment, you will have to download the JavaScript APIs with all its files and store them locally on your development machine. The local environment is intended for experienced users only and might be difficult for non-experts to properly set up. Make sure that you have installed npm and Node.js on your system before starting. Node.js and npm can be downloaded here. To download the APIs and start developing, follow these steps:

- Open your application from the list of applications and click Download API
- Unzip the file on your computer and notice the file structure of the API:



- Open the api directory with your Command-line interface, and type `npm install` to install all required node packages
- Open the `tiles.js` file in the templates directory and change the following:
 - `{{tilesLibHolder}}` - replace with `'../api'`
 - `{{userNameHolder}}` - replace with application owner of the target application
 - `{{appNameHolder}}` - replace with name of the target application
 - `{{ipAddressHolder}}` - replace with IP address of TILES Cloud server

If everything went well, you will now be ready to start developing your application. With the Local development environment, you will be writing your application code in JavaScript (similar to Cloud development environment). Once you have downloaded the API, you are free to select any IDE you want for writing the code. Using this development environment requires more set up as detailed above, but once it is properly configured you will have access to the source code of the entire TILES JS API, and you will be able to integrate your TILES application into any application running Node.js.

Rule Engine development environment

Similarly to the Cloud development environment, the Rule Engine is hosted in the TILES Cloud web portal with no need for any local setup. When selecting your app from the list of applications you will notice some differences in the application page from the Cloud and Local Development Environments. With this environment you do not need to open an external IDE or download the APIs and run them locally, but you will be able to develop your application directly in the application page. Please see Rule Engine API for instructions on how to develop applications with the Rule Engine Development Environment.

The screenshot displays the TILES Cloud web interface. At the top, a navigation bar contains the TILES Cloud logo and menu items: Home, Users, Applications (highlighted), Primitives, and Docs. Below the navigation, the application name 'garage_control_system' is shown in large red text, accompanied by a 'Start application' button. A user profile for 'Anders' is visible. The main content area is divided into several sections: 'Development environment' with an 'IFTTT key' and an 'Edit' button; 'Rule engine' with an 'Edit' button; 'List of items' with the message 'No item registered for this application.' and a '+ new item' button; 'IFTTT rules' with the message 'No IFTTT rule registered for this application.' and a '+ new IFTTT rule' button; and 'Tile rules' with the message 'No Tile rule registered for this application.' and a '+ new Tile rule' button.

Figure B-5, Rule Engine Development Environment

TILES Square Primitives

The TILES toolkit enables the definition, implementation and manipulation of data type primitives. The TILES Square primitives allow to abstract low-level implementation details into simple primitives for bridging the digital and the physical world. The primitives are implemented in the firmware of the TILES Squares, and events containing the identifier of the primitives will traverse the layers of the toolkit infrastructure upon interacting with the Squares of an application. The TILES toolkit differentiate between two types of primitives:

1. input primitives - *originating from the TILES Squares*
2. output primitives - *targeting the TILES Squares*

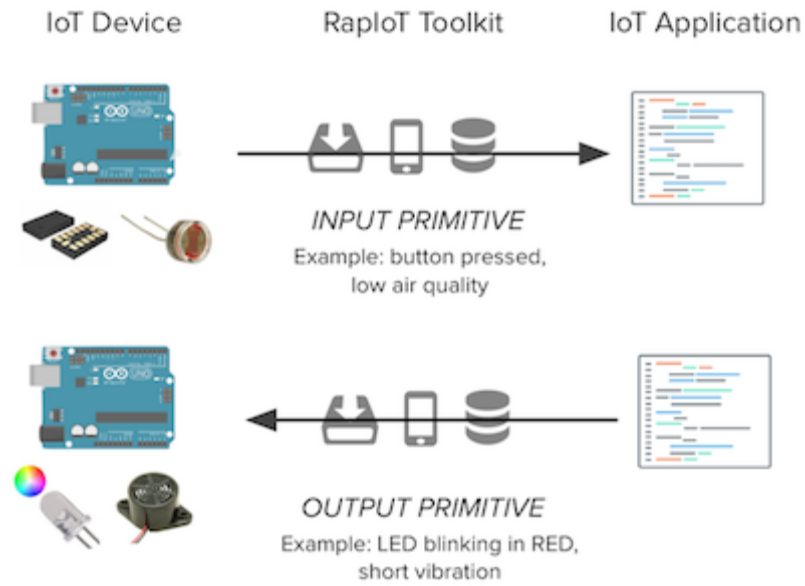


Figure B-6, RapIoT toolkit example primitives

Example of primitives supported by the TILES Squares are listed in the table below. In addition, the Squares features custom ports to be used for extending the hardware with primitives of your own design. To extend the TILES Square primitives, see the TILES toolkit Extension Process. For a complete overview of all the currently supported interaction primitives, see the list of primitives registered in the TILES Cloud web portal.

INPUT PRIMITIVES:		
Primitive	Degrees of freedom	Example mapping
Touch/tap	Single, double	Send a command, log a quantity
Tilt	Boolean (tilted/not tilted)	Select a function, binary switch
OUTPUT PRIMITIVES:		
Primitive	Degrees of freedom	Example mapping
LED feedback	Color, blink, fade (hexadecimal color)	Continuous notification about the status of process
Haptic feedback	Vibration pattern (burst, long)	Discrete notification about the status of process

Table B-1, TILES Square interaction primitives

B.2 App. Development Process

0. Process Description

If you are looking for a way to rapidly develop and prototype your IoT application ideas based on everyday object augmentation, you should keep on reading this section about the TILES toolkit Application Development Process (TADP). The TADP is a process intended to guide IoT application developers through the required steps of

prototyping a fully functional IoT application using the TILES toolkit. Whether you are a non-expert or an expert user, this section will enable you to rapidly transform your IoT application ideas into working prototypes with hardware components, message events and running program code in no time.

An illustration of the TADP and a short description of the defined steps can be seen below. Go through each subsection of the documentation sequentially and you will have a working prototype of your IoT application operable in just a few moments. Developing IoT applications have never been easier!

#	Name	Description
1	<u>Ideation Phase</u>	entails using the TILES Cards and ideation process to develop an IoT application idea.
2	<u>Create User</u>	entails creating a user in the TILES Cloud web portal.
3	<u>Create Application</u>	involves defining an application name and create the application context and selecting development environment in the TILES Cloud web portal.
4	<u>List Physical Objects</u>	entails using the TILES Cards to identify the physical objects needed in the application, and listing them by configuring the application from step 3 with items for your physical things.
5	<u>Launch Development Environment</u>	entails starting up the selected development environment and navigating to the starting point of the environment.
6	<u>Code Application</u>	involves coding the program behavior by mapping the TILES cards into executable program code with the steps listed below. <ul style="list-style-type: none"> a. Map HUMAN ACTIONS cards with TILES API events b. Map FEEDBACK cards with TILES API commands c. Map SERVICES cards with TILES API sources d. Use TILES API for additional application behavior
7	<u>Test Application</u>	involves starting application and using gateway to discover and use the TILES Squares in your application by following the steps below. <ul style="list-style-type: none"> a. Run application in test mode b. Procure and ready TILES Squares c. Procure and ready Physical Objects d. Open TILES Gateway app and log in e. Select the application in TILES Gateway f. Pair TILES Squares with items g. Use the application
8	<u>Iterate step6-7</u>	entails looping through step 6 and step 7 until the desired application behavior is accomplished.

Table B-2, Steps of TADP

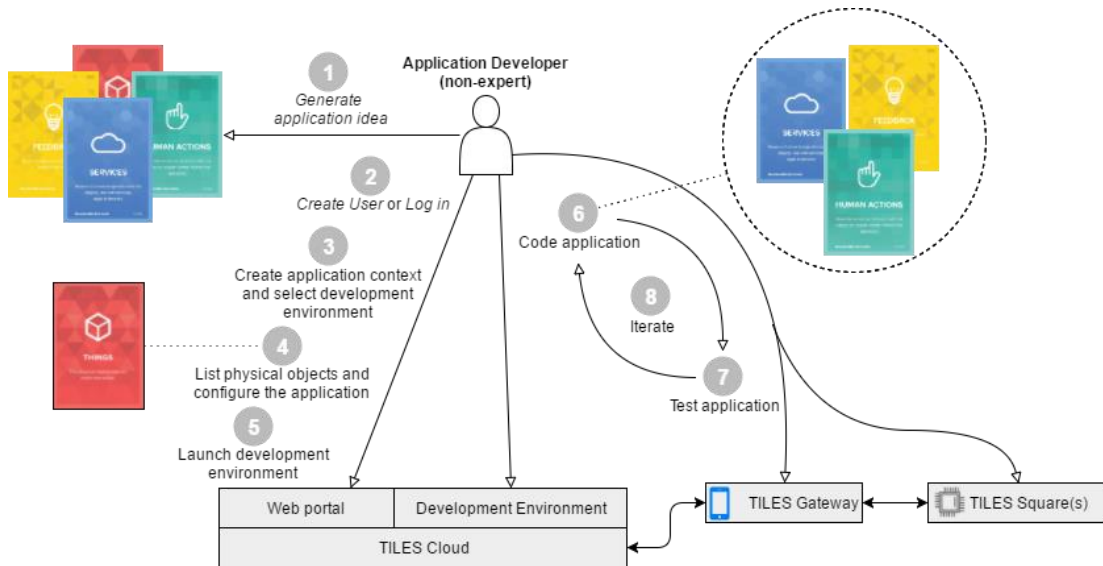


Figure B-7, TILES toolkit Application Development Process (TADP)

1. Ideation Phase

In this initial step of the process, you will be using the TILES Card game to develop your own application idea. If you have already done this, you can continue with the next step of the process.

For detailed instructions on how to use the TILES Card game, please visit the homepage of the TILES toolkit



Figure B-8, TILES Cards

2. Create User

In this step of the TILES toolkit Application Development Process you will be creating a user account in the TILES Cloud web portal. If you have already created an account, you can skip to the next step.

Creating a user account

To create a user account, navigate to the Users page, and enter a username in the form and click Post as seen in the image below.

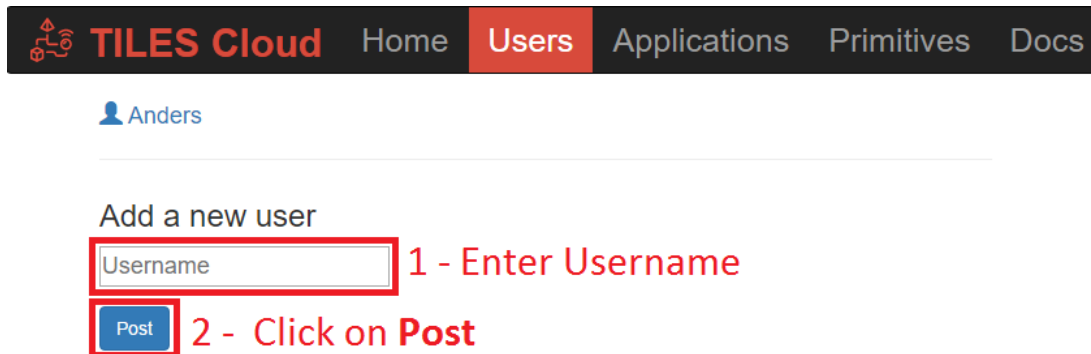


Figure B-9, Creating a new user account

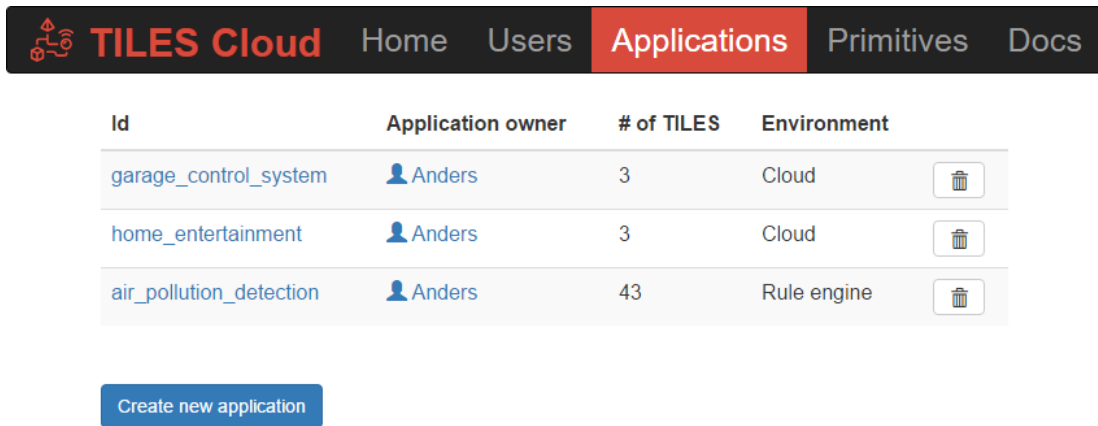
Once you have created a new user it will appear in the list. If you click on your username, you will be able to see a list of all your connected TILES Squares. If you just created a new account, this list will be empty, so don't worry if you can't see your TILES there yet.

3. Create Application

This step of the process involves defining an application name and create the application context by selecting a development environment in the TILES Cloud web portal. Which development environment you select depends on how you want to develop your application. See the development environment section to read more about the available environments. If this is the first time you use the TILES toolkit for your IoT application prototyping, we suggest you select the Cloud development environment as this is what we will be using throughout this tutorial.

Creating a new application in TILES Cloud

To create a new application, navigate to the applications page of the TILES Cloud web portal. Here you will be able to see all registered applications together with some information about them, as seen in the image below.

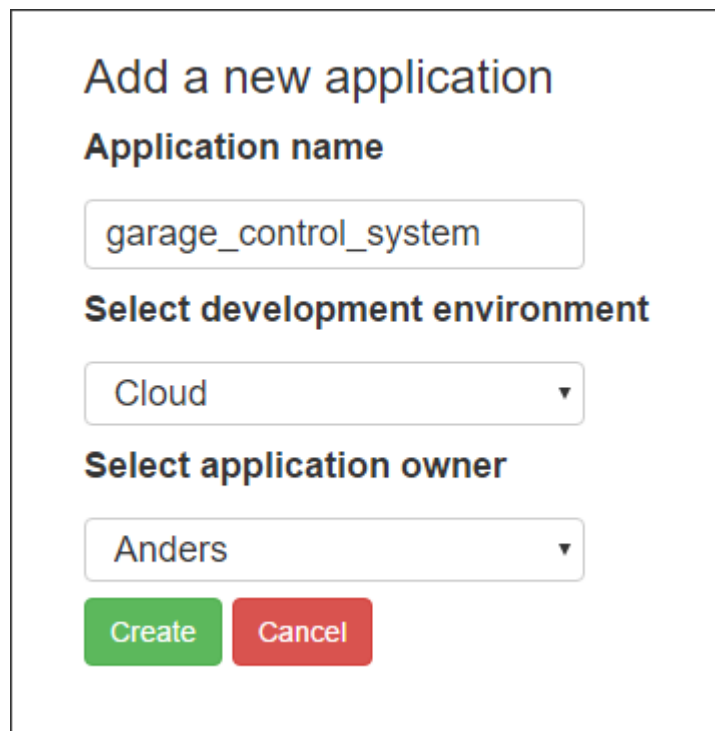


Id	Application owner	# of TILES	Environment
garage_control_system	Anders	3	Cloud
home_entertainment	Anders	3	Cloud
air_pollution_detection	Anders	43	Rule engine

Create new application

Figure B-10, List of Applications in TILES Cloud web portal

By clicking the Create new application you will be prompted with the create application form, as seen below. Here you will enter a name for your application, select the development environment, and select the owner of the application. If you are unsure of which development environment to use, we suggest the Cloud development environment. As application owner, please select your username, as you defined in the previous step. After filling the form, hit the Create button, and your application will appear in the list of applications.



Add a new application

Application name

garage_control_system

Select development environment

Cloud

Select application owner

Anders

Create Cancel

Figure B-11, Add application form in TILES Cloud web portal

4. List Physical Objects

In this step of the process you will be identifying the physical objects of your application and registering them in the application created in the previous step of the

process. This page will guide you through the step of using the TILES Cards to identify the physical objects.

a. Identifying physical objects

Using the TILES Cards makes it very easy to identify the physical objects of your application. In fact, every THINGS card used in the ideation process represents one physical object in your application. To transition from the TILES Cards you will need to write down all the THINGS Cards you have used on a piece of paper.

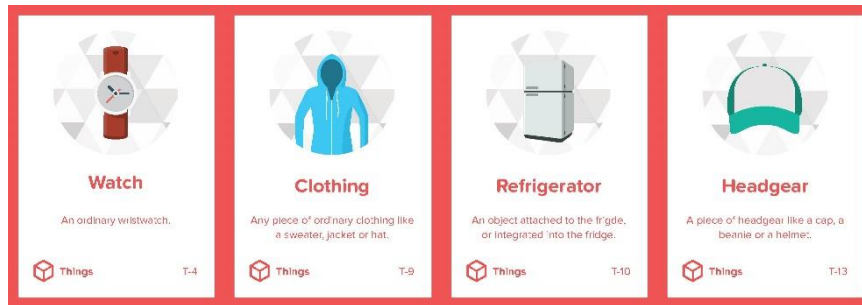


Figure B-12, THINGS Cards from TILES Cards

b. Configure items in your application

Now that you have identified all physical objects to be used in your application it is time to configure items in your application. To do this you need to navigate to the applications page and select your application from the list. If you cannot find your application in the list, you will need to go through the Create Application step of the process. After you have selected your application from the list, you will see a similar page as in the image below.

garage_control_system

Start application

Anders

Development environment
Cloud

Environment actions
Start hosting

List of items

Item name	TILE ID	Active	State	Timestamp	
front_door	unpaired	unpaired	unpaired	unpaired	
garage_door	unpaired	unpaired	unpaired	unpaired	

+ new item

Figure B-13, Application details, with two items

The `garage_control_system` application seen above already has two items registered; `front_door` and `garage_door`. By clicking the + new item button, you will be able to add a new item. Make sure to do this for all the physical objects you identified previously. You should give the items short names as you will be using these names later in your program code.

5. Launch Dev. Env.

In the previous step you configured your application with the digital placeholders for the physical objects you will use in your application. Now you are ready to start code your application, but before you can do that, you need to start the development environment in which you will be writing the application code. In this section we will show how to start the Cloud Development Environment. Please visit the development environment section for more information on other available development environments.

Starting Cloud Development Environment

To start the Cloud Development Environment, there is only one thing you need to do. Go to the application details by selecting your application from the list of applications. Here you should see a button with the text `Start hosting`. Please click this button, and your Cloud Development Environment will be started. After clicking the button, a second button will appear with the text `Launch Environment`. Click this button, and the Cloud Development Environment will be opened in a new tab in your browser. The environment should resemble the image seen below.

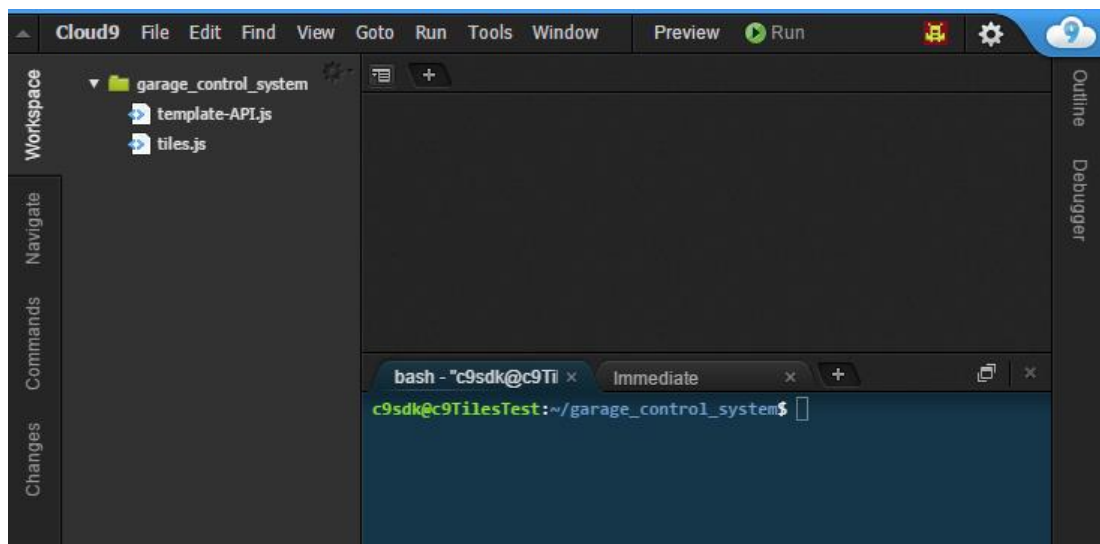


Figure B-14, Cloud Development Environment

6. Code Application

In this step of the process you will be writing the actual program code of your application. This is the most complex step of the process and has been divided into four substeps for you to follow. Do not worry if this is your first time writing program

code as this documentation will guide you through every step of the development process. The four steps are:

- a. Map HUMAN ACTIONS cards with TILES API events
- b. Map FEEDBACK cards with TILES API commands
- c. Map SERVICES cards with TILES API sources
- d. Use TILES API for additional application behavior

For this step you will be using a separate part of the documentation. When you have finished the program code you should return to this section and continue with the Application Development Process.

Cloud Development Environment

For this part, you will be writing your program code in JavaScript. At this point you should have the Cloud Development Environment running as described in the previous step of the process.

The JavaScript API will guide you through the steps of prototyping your IoT application.


7. Test Application

If you have reached this step of the process you are ready to test your application. You should now have followed all six previous steps of the application development process, including going through the JavaScript API section for instructions on how to code your application. Please go through the following steps to test your application.

a. Run application in test mode

This step is intended for Cloud Development Environment only!

To run your application in test mode using the Cloud Development Environment follow these steps:

1. Open the Cloud Development Environment as seen on the image below
2. Select the tiles.js template file by double clicking on it in the list to the left
3. Locate and click the  button
4. Notice the output text in the bash section in the bottom of the window.

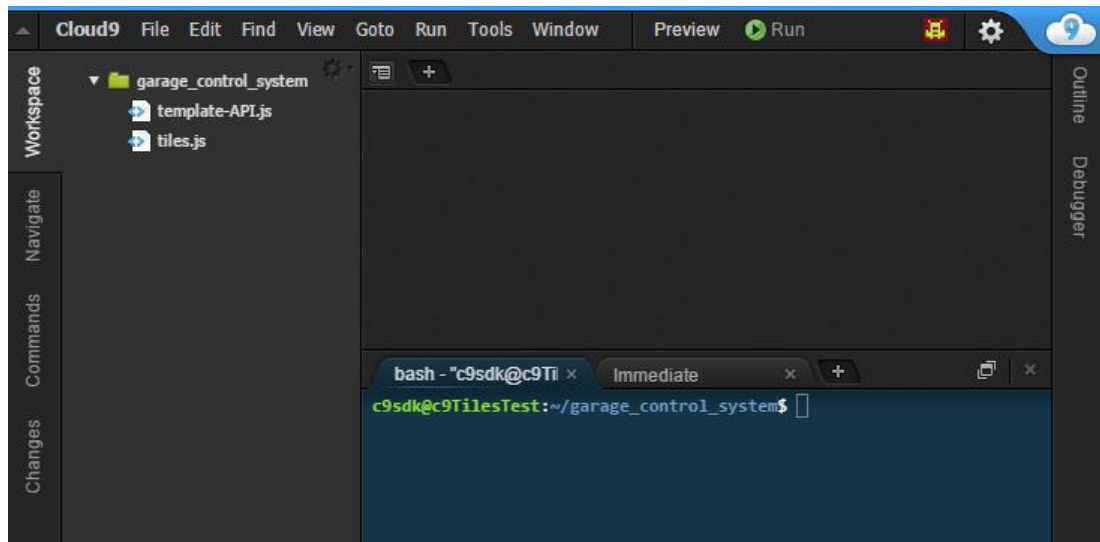


Figure B-15, Cloud Development Environment

b. Procure and ready TILES Squares

For each of the items you configured in your application in step 4 of the process, you should procure one physical TILES Square. Every item will soon be paired with the physical squares in your application.

c. Procure and ready physical objects

Similarly, for each physical TILES Square you need the physical object on which you will attach the TILES Square devices. You should now procure all the physical objects that will be used in your application. The TILES Squares will detect the HUMAN ACTIONS you perform on the objects, and pass the information as events into the application code you have just written.

d. Open TILES Gateway app and log in

1. Open up the TILES Gateway
2. Enter your username
3. Enter the host address to the TILES Cloud server
4. Enter the port number (should be 8080)
5. Click Log in!

Login

Username Anders

Host 178.62.99.218

Port 8080

Remember me

[LOG IN!](#)

Figure B-16, Log into app with server address, username and port number

e. Select the application

Select your application from the list of applications by clicking VIEW APP

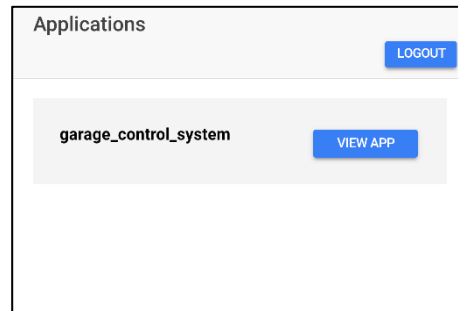


Figure B-17, List of available applications in gateway

f. Pair TILES Squares with items

After you have selected your application, you should see a list of all the items you have defined for your application. In the image below we can see that we have selected the garage_control_system that has three items defined: garage_door, front_door and car.

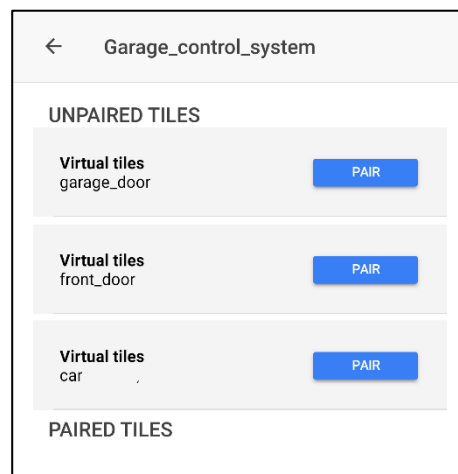


Figure B-18, List of items in application

For each of the items, click the PAIR button, and select an available physical TILES Square. The item should now move from the list UNPAIRED TILES to the list PAIRED TILES, and you have successfully paired the item and physical TILES Square. Last, but not least, you should attach the physical TILES Square to the physical objects.

g. Test/use the application

Now you should be able to use the application by interacting with the physical objects that you have attached the TILES Squares to. In addition, you will be able to get more debug information by looking at the bash section, described in step "a" above.

8. Iterate step 6-7

In order to reach the desired behavior of your application, you will probably need to iterate over step 6 and step 7 many times. After going through step 7 for the first time you most likely notice some inconsistency with your application. Write these down, and go back to step 6 to fix the errors. Once you are confident that the errors have been fixed, you can revisit step 7 and test your application again. This time it should be easier to follow the steps of testing. If you are using the same gateway app it will remember your previous configuration, and you can skip directly to the last step g. Test/use the application. Again you might notice inconsistency with the behavior of your application and you will make another iteration to fix it.

If you are satisfied with how your application behaves;

Congratulations! You have just created an IoT application prototype supporting augmentation of physical objects!

The only thing that remains is to navigate to the application page by selecting your application from the list of applications and click the Start application button, next to the name of the application. This will start the application for production.

B.3 JavaScript API

1. Introduction

If you are using either the Cloud or Local Development Environment, you will be writing your application code in JavaScript by using the TILES JavaScript API. This chapter of the documentation will guide you through the steps of transitioning from your TILES Cards ideation into a working prototype of your application. The steps of this process are listed below.

- a. Map HUMAN ACTIONS cards with TILES API events
- b. Map FEEDBACK cards with TILES API commands
- c. Map SERVICES cards with TILES API sources
- d. Use TILES API for additional application behavior

If you are already familiar with the template files of the JavaScript API you can skip to the next section of the documentation to get started with developing your application, otherwise we suggest that you finish this introduction of the JavaScript API.

After you have finished the JavaScript API chapter, you should continue with the Test Application step of the Application Development Process to test your application.

The template files

Before we start writing the application program code, let's look at the template files available with the JavaScript API. You should notice that there are two template files available in your development environment:

- tiles.js
- example-API.js

The example-API.js file is a file containing some code samples and sample usage of the JavaScript API. You can use this file to study the syntax and structure of the JavaScript API method calls.

The tiles.js, seen in the code snippet below, is an empty template file where you will be writing your own application logic. Below the code snippet we will break down the code line by line, to explain what the code does.

```

1  var tilesLib = require('{ {tilesLibHolder} }');
2
3  var client = new tilesLib.TilesClient('{ {userNameHolder} }', '{
  {appNameHolder} }', '{ {ipAddressHolder} }', 1883).connect();
4  var reader = new tilesLib.EventReader();
5  var PostmanClient = new tilesLib.PostmanClient('{enter-ip-here}',
  '{enter-port-here}');
6  var IFTTTClient = new tilesLib.IFTTTClient('{enter-ifttt-
  personal-key-here}');
7  client.on('receive', function (tileId, event) {
8      /* WORK HERE! */
9  });

```

Code snippet B-1, tiles.js template file explained

tiles.js template explained

The remaining part of this section will explain each line of the initial tiles.js template file. You can skip to the next section if you are not interested to know what the code does, or if you are eager to get started developing your own application.

```

1  var tilesLib = require('{ {tilesLibHolder} }');

```

The first line of the tiles.js code template file is a reference to the TILES JavaScript API. When you select the Cloud development environment, the reference '{ {tilesLibHolder} }' will be replaced automatically with the proper reference the API files on the Cloud development server. Using the Local development environment, you will have to manually set the reference to the downloaded API files, which will usually be located at './api'.

```

1  var client = new tilesLib.TilesClient('{ {userNameHolder} }', '{
  {appNameHolder} }', '{ {ipAddressHolder} }', 1883).connect();

```

The next line of the tiles.js template will initialize the TilesClient API. Again the 'place holder' references will be automatically configured for Cloud development environment, while needs to be manually set for Local environment. See development environment section for more details.

```
1 var reader = new tilesLib.EventReader();
2 var PostmanClient = new tilesLib.PostmanClient('{enter-ip-here}',
  '{enter-port-here}');
3 var IFTTTClient = new tilesLib.IFTTTClient('{enter-ifttt-
  personal-key-here}');
```

The next three lines will reference three additional JavaScript APIs. How to use these APIs are covered in the following three sections of this documentation: Event Reader API, Postman Client API and IFTTT Client API.

```
1 client.on('receive', function (tileId, event) {
2     /* WORK HERE! */
3 });
```

The final lines of the tiles.js template file is where you will write your application logic. The TILES infrastructure is event driven, which means that code is executed only when events are received and processed. Inside the client.on('receive,...'); code block, we will be able to determine what type of event has occurred in order to implement the reaction to the event. This is covered in more details in the Event Reader API section.

2. EventReader API

The EventReader API can be used to map HUMAN ACTIONS cards and FEEDBACK cards into TILES events and TILES commands by processing the input/output primitives for you. This section will explain how the EventReader API can be used, while the following sections will detail, with examples, how to map the HUMAN ACTIONS cards and FEEDBACK cards into JavaScript code using the EventReader API.

Initializing EventReader API

Initializing the EventReader API in your application code is done with the following lines of code.

```
1 var tilesLib = require('/tiles-lib/api');
2 var reader = new tilesLib.EventReader();
```

Code snippet B-2, Initializing EventReader API

Event Reader API usage

The technical details of using the EventReader API is explained below.

EventReader

Methods:

Name	Return value	Description
readEvent(event, client)	Tile	Passing in the 'event' and 'client' will give a Tile object (described below) which simplifies reading events.
getTile(String, client)	Tile	Passing in the 'name' of the Tile and 'client' will give a Tile object (described below) which simplifies interacting with Tiles.

Example:

```

1  var reader = new tilesLib.EventReader();
2  client.on('receive', function (tileId, event) {
3
4      var tileEvent = reader.readEvent(event, client);
5
6      var tileA = reader.getTile('Tile_c5', client);
7  });

```

Code snippet B-3, EventReader API example

Tile

Properties:

Name	Type	Description
name	String	Holds the name of the Tile.
id	String	Holds the unique ID of the Tile.
isDoubleTap	Boolean	Holds the 'double tap' state of the Tile when reading an event. This will be true if the received event is a 'double tap'.
isSingleTap	Boolean	Holds the 'single tap' state of the Tile when reading an event.
isTilt	Boolean	Holds the 'tilt' state of the Tile when reading an event.

Methods:

Name	Description
hapticBurst()	This will send a burst command to the Tile, which will make the Tile vibrate with a burst pattern.
hapticLong()	This will send a vibrate command to the Tile, which will make the Tile vibrate for a moment.
ledBlink(String)	This will send a blink command to the Tile, with a color string. This will make the Tile blink with the respective color.
ledOn(String)	This will send an LED-on command to the Tile, which will turn the LED on with the respective color.
ledOff()	This will send an LED-off command, turning the LED of the Tile off.

Example:

```

1  var reader = new tilesLib.EventReader();
2  client.on('receive', function (tileId, event) {
3      var tileEvent = reader.readEvent(event, client);
4      var tileA = reader.getTile('Tile_c5', client);
5
6      if (tileEvent.name == tileA.name) { /* Check if the event
7  originates from 'tileA' */
8          tileA.ledOn('FF00FF'); /* Turn on LED on tileA to color
9  #FF00FF */
10         }
11         if (tileEvent.isSingleTap) { /* Check if the event is a
12         'single tap' event */
13             tileEvent.hapticBurst(); /* Vibrate the tile that was
14             tapped */
15         }
16     });

```

Code snippet B-4, Tile methods example

TILES Client API usage

If you are an expert user, and you need to send custom messages to the TILES Squares, you can use the TILES Client to achieve this.

TilesClient

Methods:

Name	Description
send(String, String, String, String)	Using the send method, you can send a command to a TILES Square.

Example:

```

1  var client = new tilesLib.TilesClient('Anders',
    'garage_control_system', '178.62.99.218', 1883).connect();
2  client.on('receive', function (tileId, event) {
3      /* To turn on the led to color red on TILES Square with name
        'tile_c5' */
4      client.send('tile_c5', 'led', 'on', 'red')
5  });

```

Code snippet B-5, TileClient example

3. Map HUMAN ACTIONS

The previous section introduced the EventReader API for processing events and commands in the JavaScript API. This section will go more into details on how to use the EventReader API to map the HUMAN ACTIONS cards to the TILE events in your application code.

Example 1: Tap Watch

This example shows how THINGS Watch and HUMAN ACTIONS Tap is mapped to JavaScript code.



Figure B-19, Tap Watch

Looking at the code snippet below, we see that the watch is defined in the JavaScript code in line 4. In line 6 we see how we detect the origin of the event (from 'Watch'), and in line 7 we see how the nature of the event (double tap) is detected.

```

1  var reader = new tilesLib.EventReader();
2  client.on('receive', function (tileId, event) {
3      var tileEvent = reader.readEvent(event, client);
4      var watch = reader.getTile('watch', client);
5      if (tileEvent.name == watch.name) {
6          /* Check if the event originates from 'watch' */
7          if (tileEvent.isDoubleTap) {
8              /* Check if the event is a double tap event */
9              /* Write your FEEDBACK here */
10         }
11     }
12 });

```

Code snippet B-6, Tap Watch example

Example 2: Tilt Headgear

This example shows how THINGS Headgear and HUMAN ACTIONS Tilt is mapped to JavaScript code.

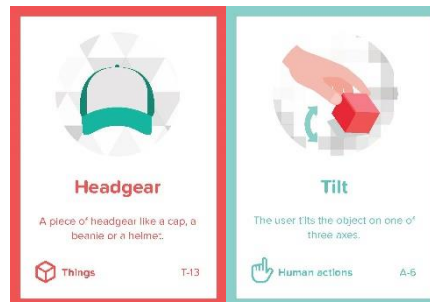


Figure B-20, Tilt Headgear

Looking at the code snippet below, we see that the headgear is defined in the JavaScript code in line 4. In line 6 we see how we detect the origin of the event (from 'Headgear'), and in line 7 we see how the nature of the event (tilt) is detected.

```

1  var reader = new tilesLib.EventReader();
2  client.on('receive', function (tileId, event) {
3    var tileEvent = reader.readEvent(event, client);
4    var headgear = reader.getTile('headgear', client);
5    if (tileEvent.name == headgear.name) {
6      /* Check if the event originates from 'headgear' */
7      if (tileEvent.isTilt) {
8        /* Check if the event is a tilt event */
9        /* Write your FEEDBACK here */
10     }
11   }
12 });

```

Code snippet B-7, Tilt Headgear example

4. Map FEEDBACK

Section 2. EventReader API covered how to use the EventReader API for processing events and commands in the JavaScript API. The previous section covered how to map the HUMAN ACTIONS cards into TILES events. In this section we will be looking into how to map the FEEDBACK cards into TILES commands. You have probably realized by now that the TILES events are input primitive messages from the TILES Squares to your code, while the TILES commands are output primitive messages from your code to the TILES Squares.

Example 1: Change color on Plant

This example shows how FEEDBACK Color change and THINGS Plant is mapped to JavaScript code. This is useful if you want the color on your plant to change.



Figure B-21, Change color on Plant

Looking at the code snippet below, we see that the plant is defined in the JavaScript code in line 3. The code for sending the change color command can be seen in line 5. Here we are changing the color to 'FF00FF' or pink.

```

1  var reader = new tilesLib.EventReader();
2  client.on('receive', function (tileId, event) {
3      var plant = reader.getTile('plant', client);
4      plant.ledOn('FF00FF');
5  });

```

Code snippet B-8, Change color on Plant example

Example 2: Vibrate Refrigerator

This example shows how FEEDBACK Vibrate and THINGS Refrigerator is mapped to JavaScript code. This is useful if you want your refrigerator to vibrate.



Figure B-22, Vibrate Refrigerator

Looking at the code snippet below, we see that the refrigerator is defined in the JavaScript code in line 3. The code for sending the vibrate command can be seen in line 5. Here we are vibrating the refrigerator with short bursts.

```

1  var reader = new tilesLib.EventReader();
2  client.on('receive', function (tileId, event) {
3      var refrigerator = reader.getTile('refrigerator', client);
4      refrigerator.hapticBurst();
5  });

```

Code snippet B-9, Vibrate Refrigerator example

5. Map SERVICES

Mapping SERVICES cards into TILES sources is a tricky concept and it requires knowledge of external web services. If don't know any web services it is recommended that you drop the SERVICES cards from your application. If you want to learn more about web services it is recommended that you check out IFTTT.

Most of the SERVICES cards can be realized by creating applets in IFTTT and connect them with your TILES application.

The following sections will introduce two TILES JavaScript APIs that can be used to map SERVICES cards into TILES sources.

Postman Client API

The Postman Client API is an API that will make it possible to send GET and POST request to specific IP-addresses. This API is intended to bridge your application with other web services you need to integrate into your IoT application. The PostmanClient takes two inputs upon initialization, IP-address of the web server and the port number of the web services.

```
1 var tilesLib = require('/tiles-lib/api');
2 var PostmanClient = new tilesLib.PostmanClient('{enter-ip-here}',
  '{enter-port-here}');
```

Code snippet B-10, Initialize PostmanClient API

Postman Client API usage

PostmanClient

Methods:

Name	Description
get(String)	Will send a HTTP GET request to the input String URL.
post(String, function)	Will send a HTTP POST request to the input String URL, and will return the response to the callback function .

Example:

```
3 var reader = new tilesLib.EventReader();
4 var PostmanClient = new tilesLib.PostmanClient('192.168.1.111',
  8080);
5 client.on('receive', function (tileId, event) {
6   var tileEvent = reader.readEvent(event, client);
7   if (tileEvent.isSingleTap) {
8     PostmanClient.get('url-name');
9     PostmanClient.post('url-name', function (response) {
10      console.log(response);
11    });
12  }
13 });
```

Code snippet B-11, PostmanClient example

IFTTT Client API

The IFTTT Client API is a specialized version of the PostmanClient API, and will enable you to use IFTTT to trigger a wide variety of popular web services with only a few lines of code. For this API you must create an account at IFTTT, and configure the maker channel to get your own personal key. This key must be used as input to the IFTTT Client API constructor as seen below.

```
1 var tilesLib = require('/tiles-lib/api');
2 var IFTTTClient = new tilesLib.IFTTTClient('{enter-ifttt-
  personal-key-here}');
```

Code snippet B-12, Initialize IFTTTClient API

IFTTTClient API usage

IFTTTClient

Methods:

Name	Description
send(String, String, String, String)	Will send a POST request to the target 'trigger name'. The first input parameter is the 'trigger name' configured in the IFTTT applet, while the three next are optional parameters to be used in the IFTTT applet.

Example:

```
1 var reader = new tilesLib.EventReader();
2 var IFTTTClient = new tilesLib.IFTTTClient('-MXtrSsdb-WQxdmbW-
  CuA');
3 client.on('receive', function (tileId, event) {
4   var tileEvent = reader.readEvent(event, client);
5   if (tileEvent.isTilt) {
6     IFTTTClient.send('send_email');
7     IFTTTClient.send('lights_on', 'FF00FF', 21, '08:00');
8   }
9 });
```

Code snippet B-13, IFTTTClient example

Example: Tilt Watch to send Email

This example shows how SERVICES Mail is mapped to TILES source using IFTTT. The THINGS Watch and HUMAN ACTIONS Tilt is used to show a complete scenario example.



Figure B-23, Tilt Watch to send Email (using IFTTT)

This sample is a bit complicated and is intended for you if you are familiar with IFTTT or web services. To make this work you will need to set up your applet using the IFTTT maker channel manually.

In this example we have preconfigured an IFTTT maker channel with the private key -MXtrSsdb-WQxdmbW-CuA, and created an applet with trigger name `send_email` that accepts two custom arguments email address and email message.

```

1  var reader = new tilesLib.EventReader();
2  var IFTTTClient = new tilesLib.IFTTTClient('-MXtrSsdb-WQxdmbW-
   CuA');
3  client.on('receive', function (tileId, event) {
4      var tileEvent = reader.readEvent(event, client);
5      var watch = reader.getTile('watch', client);
6      if (tileEvent.name == watch.name) {
7          if (tileEvent.isTilt) {
8              IFTTTClient.send('send_email',
9                  'tiles@tilestoolkit.io', 'This is an email');
10         }
11     });

```

Code snippet B-14, Tilt Watch to send Email example

6. Example Scenario 1




Description:

1. Single tap on 'Eyewear' will set color on 'Eyewear' from looping array [red, green, blue, white, pink].
2. Double tap on 'Eyewear' will vibrate 'Headgear' AND set color on 'Headgear' to the same as 'Eyewear'.
3. Tilting 'Eyewear' will turn off LED on 'Headgear'.





Example TILES Cards configuration:

This is just one specific TILES Cards configuration. There are countless ways to explain this scenario with the TILES Cards. You should use the TILES Cards just the way that feels natural to you.




1.

 <p style="text-align: center;">Tap</p> <p style="font-size: 8px;">The user taps the object, either with a single tap or double tap.</p> <p style="font-size: 8px;">Human actions A-B</p>	 <p style="text-align: center;">Eyewear</p> <p style="font-size: 8px;">A pair of ordinary glasses or sunglasses.</p> <p style="font-size: 8px;">Things T-5</p>	 <p style="text-align: center;">Eyewear</p> <p style="font-size: 8px;">A pair of ordinary glasses or sunglasses.</p> <p style="font-size: 8px;">Things T-5</p>	 <p style="text-align: center;">Color change</p> <p style="font-size: 8px;">A light on the object changes from one color to another.</p> <p style="font-size: 8px;">Feedback F-6</p>
--	---	---	--

2.

 <p style="text-align: center;">Tap</p> <p style="font-size: 8px;">The user taps the object, either with a single tap or double tap.</p> <p style="font-size: 8px;">Human actions A-B</p>	 <p style="text-align: center;">Eyewear</p> <p style="font-size: 8px;">A pair of ordinary glasses or sunglasses.</p> <p style="font-size: 8px;">Things T-5</p>	 <p style="text-align: center;">Headgear</p> <p style="font-size: 8px;">A piece of headgear like a cap, a beanie or a helmet.</p> <p style="font-size: 8px;">Things T-13</p>	 <p style="text-align: center;">Color change</p> <p style="font-size: 8px;">A light on the object changes from one color to another.</p> <p style="font-size: 8px;">Feedback F-6</p>	 <p style="text-align: center;">Vibrate</p> <p style="font-size: 8px;">The object starts vibrating.</p> <p style="font-size: 8px;">Feedback F-8</p>
--	---	---	--	--

3.

 <p style="text-align: center;">Tilt</p> <p style="font-size: 8px;">The user tilts the object on one of three axes.</p> <p style="font-size: 8px;">Human actions A-5</p>	 <p style="text-align: center;">Eyewear</p> <p style="font-size: 8px;">A pair of ordinary glasses or sunglasses.</p> <p style="font-size: 8px;">Things T-5</p>	 <p style="text-align: center;">Headgear</p> <p style="font-size: 8px;">A piece of headgear like a cap, a beanie or a helmet.</p> <p style="font-size: 8px;">Things T-13</p>	 <p style="text-align: center;">Color change</p> <p style="font-size: 8px;">A light on the object changes from one color to another.</p> <p style="font-size: 8px;">Feedback F-6</p>
--	--	--	---

JavaScript code:

```

1  var tilesLib = require('/tiles-lib/api');
2  var client = new tilesLib.TilesClient('Anders', 'Scenario1',
   '138.68.144.206', 1883).connect();
3  var reader = new tilesLib.EventReader();
4  var colors = ['red', 'green', 'blue', 'white', 'FF00FF'];
5  var ct = 0;
6  client.on('receive', function (tileId, event) {
7      var tileEvent = reader.readEvent(event, client);
8      var eyewear = reader.getTile('eyewear_tile', client);
9      var headgear = reader.getTile('headgear_tile', client);
10     /* First instruction (first set of cards) */
11     if (tileEvent.name === eyewear.name && tileEvent.isSingleTap)
12     {
13         ct++;
14         if (ct >= colors.length) ct = 0;
15         eyewear.ledOn(colors[ct]);
16     /* Second instruction (second set of cards) */
17     } else if (tileEvent.name === eyewear.name &&
18         tileEvent.isDoubleTap) {
19         headgear.ledOn(colors[ct]);
20         headgear.hapticBurst();
21     /* Third instruction (third set of cards) */
22     } else if (tileEvent.name === eyewear.name &&
23         tileEvent.isTilt) {
24         headgear.ledOff();
25     }
26 });

```

Code snippet B-15, JavaScript example scenario 1

7. Example Scenario 2**Context:**

You are working in an office and you have decided to make your work station smart. You want to be able to turn on and off the lights from your desk, without getting up and hit the light switch on the other side of the room. In addition, you would like to start and stop the music on the stereo on the other side of the room without leaving your comfortable office chair.




Description:




1. Single tap on 'Office desk' will set color of lights (trigger IFTTT applet 'set_lights' with color 'FF00FF').
2. Double tap on 'Office desk' will turn off lights (trigger IFTTT applet 'lights_off').
3. Single tap on 'Watch' will start music (trigger Postman to server '192.168.1.111' port '3000' with trigger 'start_music').
4. Double tap on 'Watch' will stop music (trigger Postman to server '192.168.1.111' port '3000' with trigger 'stop_music').



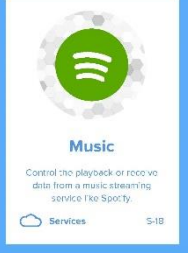
Example TILES Cards configuration:



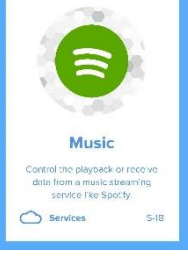
This is just one specific TILES Cards configuration. There are countless ways to explain this scenario with the TILES Cards. You should use the TILES Cards just the way that feels natural to you.

1.

 <p style="text-align: center;">Tap</p> <p style="font-size: 8px;">The user taps the object, either with a single tap or double tap.</p> <p style="font-size: 8px;">Human actions A-B</p>	 <p style="text-align: center;">Office desk</p> <p style="font-size: 8px;">Some part of your office workspace, like the desk itself or a lamp.</p> <p style="font-size: 8px;">Things T-11</p>	 <p style="text-align: center;">Custom channel</p> <p style="font-size: 8px;">If you have 'oles for other data sources or sensors, add them here.</p> <p style="font-size: 8px;">Services S-1</p>
--	--	--
2.

 <p style="text-align: center;">Tap</p> <p style="font-size: 8px;">The user taps the object, either with a single tap or double tap.</p> <p style="font-size: 8px;">Human actions A-B</p>	 <p style="text-align: center;">Office desk</p> <p style="font-size: 8px;">Some part of your office workspace, like the desk itself or a lamp.</p> <p style="font-size: 8px;">Things T-11</p>	 <p style="text-align: center;">Custom channel</p> <p style="font-size: 8px;">If you have 'oles for other data sources or sensors, add them here.</p> <p style="font-size: 8px;">Services S-1</p>
--	--	--
3.

 <p style="text-align: center;">Tap</p> <p style="font-size: 8px;">The user taps the object, either with a single tap or double tap.</p> <p style="font-size: 8px;">Human actions A-B</p>	 <p style="text-align: center;">Watch</p> <p style="font-size: 8px;">An ordinary wristwatch.</p> <p style="font-size: 8px;">Things T-4</p>	 <p style="text-align: center;">Music</p> <p style="font-size: 8px;">Control the playback or receive data from a music streaming service like Spotify.</p> <p style="font-size: 8px;">Services S-18</p>
---	--	---
4.

 <p style="text-align: center;">Tap</p> <p style="font-size: 8px;">The user taps the object, either with a single tap or double tap.</p> <p style="font-size: 8px;">Human actions A-B</p>	 <p style="text-align: center;">Watch</p> <p style="font-size: 8px;">An ordinary wristwatch.</p> <p style="font-size: 8px;">Things T-4</p>	 <p style="text-align: center;">Music</p> <p style="font-size: 8px;">Control the playback or receive data from a music streaming service like Spotify.</p> <p style="font-size: 8px;">Services S-18</p>
--	---	--

JavaScript code:

```

1  var tilesLib = require('/tiles-lib/api');
2  var client = new tilesLib.TilesClient('Anders', 'Scenario2',
   '138.68.144.206', 1883).connect();
3  var reader = new tilesLib.EventReader();
4  var IFTTTClient = new tilesLib.IFTTTClient('-MXtrSsdb-WQxdmbW-
   CuA');
5  var PostmanClient = new tilesLib.PostmanClient('192.168.1.111',
   3000);
6
7  client.on('receive', function (tileId, event) {
8      var tileEvent = reader.readEvent(event, client);
9      var officeDesk = reader.getTile('office_desk_tile', client);
10     var watch = reader.getTile('watch_tile', client);
11     /* First instruction (first set of cards) */
12     if (tileEvent.name === officeDesk.name &&
        tileEvent.isSingleTap) {
13         IFTTTClient.send('set_lights', 'FF00FF');
14         /* Second instruction (second set of cards) */
15     } else if (tileEvent.name === officeDesk.name &&
        tileEvent.isDoubleTap) {
16         IFTTTClient.send('lights_off');
17         /* Third instruction (third set of cards) */
18     } else if (tileEvent.name === watch.name &&
        tileEvent.isSingleTap) {
19         PostmanClient.get('start_music');
20         /* Fourth instruction (fourth set of cards) */
21     } else if (tileEvent.name === watch.name &&
        tileEvent.isDoubleTap) {
22         PostmanClient.get('stop_music');
23     }
24 });

```

Code snippet B-16, JavaScript example scenario 2

B.4 Toolkit Extension Process

0. Process Description

If you are looking for a way to customize the TILES toolkit by adding additional hardware features in the TILES Squares or if you want to extend the development APIs of the toolkit, then the TILES toolkit Extension Process (TEP) is intended for you. The TEP is a process intended to guide expert users through the steps of extending the TILES toolkit with new and improved hardware and software capabilities. Following the steps of the TEP should enable you to take the existing TILES toolkit, and customize it to fit your needs. Employing the TEP systematically implies that the new features of the toolkit will be available to non-expert users in IoT application development.

An illustration of the TEP and a short description of the defined steps can be seen below. Go through each subsection of the documentation sequentially and you will end up with an extended TILES toolkit in just a few moments.

#	Name	Description
1	<u>Device Development</u>	involves extending the hardware prototype, possibly with additional hardware, and implement input/output primitives in firmware.
2	<u>Library Development</u>	involves implementing new features to the toolkit APIs in order for application developers to be able to utilize the capabilities created in step 1.
3	<u>API Deployment</u>	entails deploying the extended libraries to the TILES Cloud server.
4	<u>TILES Card Deck Creation (optional)</u>	involves creating a deck of TILES Cards by removing all cards that are not supported by the extended hardware configuration of the TILES Squares and possibly creating additional cards for the new interaction primitives.

Table B-3, Steps of TEP

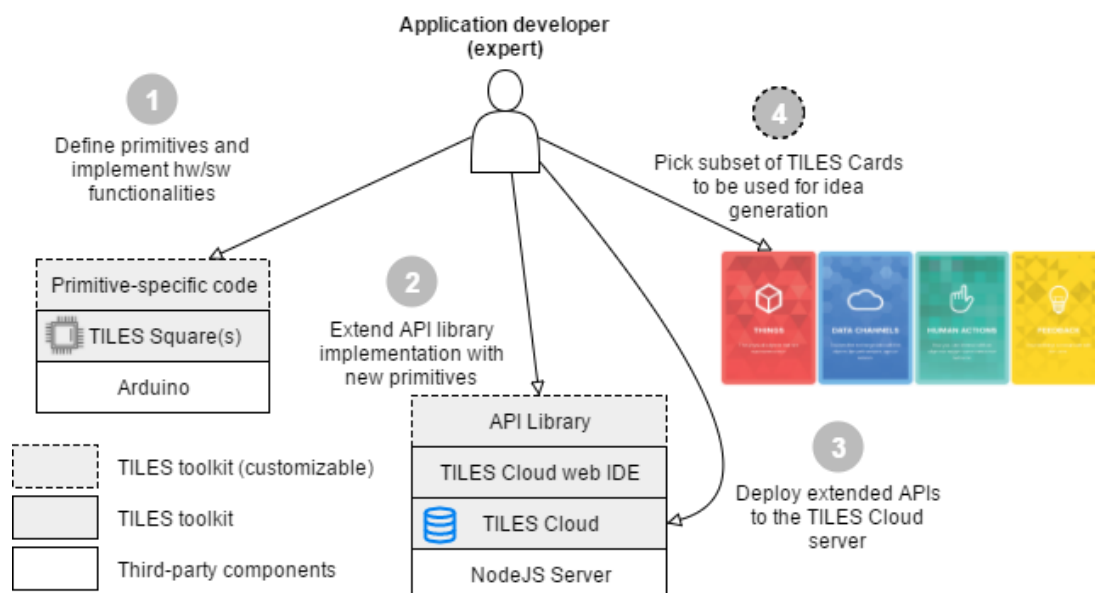


Figure B-24, TILES toolkit Extension Process

1. Device Development

This step of the TEP consist of several substeps that can be seen below. All the substeps listed below are concerned with hardware and firmware of the physical TILES Squares. The order of the steps are not important, but some of the steps depend on each other. For example, you should obviously acquire a TILES Square before you extend its hardware.

- a. Acquire TILES Squares
- b. Download and install the Arduino IDE
- c. Acquire TILES Square firmware
- d. Extend hardware of TILES Squares
- e. Extend firmware of TILES Squares
- f. Flash the firmware to your TILES Squares

At the end of this page you will find an explanation of the structure of the TILES Square firmware together with some firmware example code.

a. Acquire TILES Squares

The first thing you will need to do is to acquire the TILES Squares to be used in your customized project. Please visit the homepage of the TILES toolkit for instructions on how to acquire TILES Squares.

b. Download and install the Arduino IDE

To develop the firmware of the TILES Squares you need to download and install the Arduino IDE. Please follow the instructions on the Arduino homepage

c. Acquire TILES Square firmware

Now that you have installed the Arduino IDE and have acquired your TILES Squares, you need to get access to and download the source code of the TILES Square firmware. Please see the homepage of the TILES toolkit for instructions on how to get the firmware source code.

d. Extend hardware of TILES Squares

The next thing you will need to do is to extend the hardware of the TILES Squares. This entails plugin in the new hardware components to the I2C ports or soldering the hardware to the custom ports on the TILES Squares. Please see the hardware schematics available together with the source code of the firmware or at the homepage of the TILES toolkit.

e. Extend firmware of TILES Squares

Now that you have extended the physical hardware capabilities of the TILES Squares to fit your need, you will need to implement the capabilities in the firmware. This is not intended to teach you firmware development as you are expected to be familiar with the concept. Please visit the Arduino docs section and the RFDuino docs section for more information about firmware development.

f. Flashing the firmware to your TILES Squares

The last substep of this step of the process is to flash the new firmware to all the squares featuring the extended hardware capabilities in your customized toolkit. This flashing procedure can be done by attaching the TILES Squares to a computer running the Arduino IDE and hit the Upload button. For more information, please visit the uploading section on the Arduino IDE documentation.

TILES Square firmware structure

When you open the source code of the firmware you will notice the traditional setup-loop structure of Arduino. The setup function is called when the TILES Square is powered on and will initialize and set up all the hardware components of the Square. The loop function is called recurrently while the Square is powered on and will detect the input primitives such as tap, single tap and double tap. This is where you will be

implementing your input primitives. In addition the RFduinoBLE library is used to handle the BLE connection. When a message is sent to the TILES Square the RFduinoBLE_onReceive function will be triggered. This is where you will be implementing your output primitives.

Example output primitive

Below you can see an example of a new output primitive. This primitive will react to the led, on, {color} command and will turn the LED of the Square to the input hex color.

```
//Callback when a data chunk is received. OBS! Data chunks must be 20KB (=20 ASCII characters) maximum!
void RFduinoBLE_onReceive(char *data, int len)
{
  String command;
  command = data;
  command = command.substring(0, len);

  int commaIndex = command.indexOf(',');
  // Search for the next comma just after the first
  int secondCommaIndex = command.indexOf(',', commaIndex + 1);

  String firstValue = command.substring(0, commaIndex);
  String secondValue = command.substring(commaIndex + 1, secondCommaIndex);
  String thirdValue = command.substring(secondCommaIndex + 1);

  if (firstValue == "led") {
    if (secondValue == "on")
    {
      setColor(thirdValue);
    }
  }
}

void setColor(String color) {

  if (color.length() == 6) {
    int red, green, blue;
    parseColorString(color, red, green, blue);
    setColorRGB(red, green, blue);
  }
}

void parseColorString(String color, int& red, int& green, int& blue) {
  unsigned int colorHex = strtoul(&color[0], NULL, 16);
  red = (int)((colorHex >> 16) & 0xFF); // Extract the RR byte
  green = (int)((colorHex >> 8) & 0xFF); // Extract the GG byte
  blue = (int)((colorHex) & 0xFF); // Extract the BB byte
}

void setColorRGB(int red, int green, int blue)
{
  #ifdef COMMON_ANODE
  red = 255 - red;
  green = 255 - green;
  blue = 255 - blue;
  #endif
  analogWrite(RED_LED_PIN, red);
  analogWrite(GREEN_LED_PIN, green);
  analogWrite(BLUE_LED_PIN, blue);
}
```

Code snippet B-17, Firmware code, set output primitive LED to specific color

2. Library Development

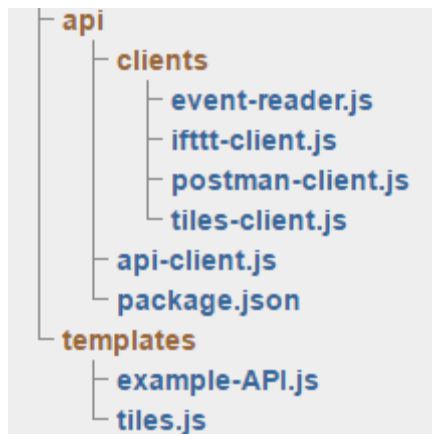
Similarly to the previous step of the TEP, this step also contains several substeps that can be seen in the list below. All the steps in the list below are concerned with the APIs of the TILES toolkit.

- a. Download JavaScript API
- b. Add detection of primitives to the EventReader client
- c. Add new clients to the JavaScript API
- d. Add new primitives to Rule Engine API

At the end of this page you will find code samples on how to add primitives to the EventReader client of the JavaScript API.

a. Download JavaScript API

The first thing you will need to do is to download the latest version of the JavaScript API. You should familiarize yourself with the structure of the API.



The clients directory contains the JavaScript client APIs. Please see the list below for a short description of their responsibilities.

- event-reader.js - EventReader client for reading events and sending commands to the TILES Squares
- ifttt-client.js - IFTTTClient for triggering applets configured in IFTTT
- postman-client.js - PostmanClient for sending web requests as GET or POST requests to any available web service.
- tiles-client.js - TilesClient is used to handle background communication with the TILES Cloud server and will be responsible for connecting to the server and forwarding the events and commands to and from the EventReader client in the background.

b. Add detection of primitives to the EventReader client

In order to make non-expert able to use the new primitives in their program code, the primitives should be registered to the EventReader client API. The input primitives should be added to the readEvent prototype function, while output primitives should

be added to the `getTile` prototype function. Below is an example of how to read the single tap event and how to send led on command.

```

1  EventReader.prototype.readEvent = function (event, client) {
2    var tile = this.getTile(event.name, client);
3    tile.isSingleTap = (event.properties[0] === 'tap' &&
4      event.properties[1].startsWith('double'));
5    return tile;
6  }

```

Code snippet B-18, read double tap in EventReader API

In line 3 of the code snippet seen above, a new property named `isSingleTap` will be added to the returned tile object. This will enable non-experts to simply check this property flag instead of using the TilesClient API to parse the received event.

```

1  EventReader.prototype.getTile = function (name, client) {
2    var id = 0;
3    if (client.tiles[name]) {
4      id = client.tiles[name];
5    }
6    var tile = {
7      name: name,
8      id: id
9    };
10   tile.ledOn = function (color) {
11     client.send(id, 'led', 'on', color);
12   }
13   return tile;
14 }

```

Code snippet B-19, getTile in EventReader API

In line 10 of the code snippet above, we can see that a new function `ledOn` is added to the returned tile object. Calling this function with a string representation of a color will call the appropriate send instruction to trigger a command to the respective TILES Square.

c. Add new clients to the JavaScript API

Optionally, if a new service is to be added to the JavaScript API, you could create a new client. Adding a new client could be useful if a new specific web service should be introduced into the JavaScript API. The PostmanClient is an example of a client API that is implemented to enable TILES application developers to use RESTful services in their application. To add a new client, simply create a new file in the clients directory and write the code to your client here. Finally, in order to make the client available in the JavaScript API, open the file `api-client.js` and add your newly created client to expose it through the JavaScript API.

d. Add new primitives to the Rule Engine API

Adding a new primitive to the RuleEngine API is very simple. All you have to do is to navigate to the primitives page in the TILES Cloud web portal and add your input/output primitive to the tables of primitives. For the input primitives, the name column is the name visible in the Rule Engine development environment so you can

pick any name you like. The property[0] and property[1] columns, however, must match the event property of the primitive. For the output primitive the name and property[0] are used to trigger the proper command, while selecting the custom property enables users to input a custom string to be sent with the command. Enabling the custom command will for instance enable users to enter a hexadecimal representation of the color they want to set to the LED of a TILES Square.

3. API Deployment

Deploying the extended API to the TILES Cloud server is achieved by following the steps listed below.

- a. Add placeholders
- b. Replace API in root lib
- c. Add zipped API to lib

a. Add placeholders

The first thing you must do before you deploy the new extended TILES API is to make sure that the placeholders are properly added to the template files of the API. Please see the code snippet below for correct placeholders. The placeholders (tilesLibHolder, userNameHolder, appNameHolder and ipAddressHolder) will be automatically replaced by the TILES Cloud server for all TILES applications on creation. This way the non-experts will not need to configure the client templates themselves before they can get started with the application coding. Additionally, the AUTO GENERATED CODE lines should be present immediately after the client.on(...){ method as seen on the code snippet. This will enable the Cloud server to automatically add template code for items of an application.

```
15 var tilesLib = require('{{tilesLibHolder}}');
16 var client = new tilesLib.TilesClient('{{userNameHolder}}',
    '{{appNameHolder}}', '{{ipAddressHolder}}', 1883).connect();
17 client.on('receive', function (tileId, event) {
18     /* AUTO GENERATED CODE START (do not remove) */
19     /* AUTO GENERATED CODE END (do not remove) */
20     ...
21 }
```

Code snippet B-20, Placeholders of template files

b. Replace API in root lib

For this step you will need access to the TILES Cloud server. On the TILES Cloud server the TILES library files are located in /tiles-lib. You will need to replace the files in this directory with the extended API you have just developed. Before you copy the files, make sure that you remove the node_modules directory. After you have replaced the API, you need to type npm install into the api directory to install all the required modules. Your extended JavaScript API is now available for the Cloud Development Environment.

c. Add zipped API to lib

In order to make the API available for developers using the Local Development Environment, you will need to zip the whole API (except the `node_modules` directory), and copy it into the `/tiles-lib` directory on the TILES Cloud server. In the zipped API you do not need the placeholders or AUTO GENERATE CODE comments from step a. as the server will not replace the code in these files. Your extended JavaScript API is now available for the Local Development Environment.

4. TILES Card Deck Creation

Even though this step is optional, it is highly recommended in order to simplify the development process for non-experts using the extended toolkit. The TILES Cards is an ideation process that has been tested and tailored to help non-expert users to develop ideas for IoT applications. Following this final step of the TEP will provide non-experts with a deck of TILES Cards that are supported by your extended TILES toolkit, and removing those cards that are not supported.

There are only two things that needs to be done in this step:

- a. Remove unsupported TILES Cards
- b. Create new TILES Cards

a. Remove unsupported TILES Cards

Starting from the original TILES Card deck you will need to remove those cards that are not supported by your extended TILES toolkit. For example, if your TILES Square firmware does not support detecting proximity, the proximity sensor card should be removed from the card deck.

b. Create new TILES Cards

After you have removed those cards that are not supported by your extended TILES toolkit, you need to create additional cards for the primitives that you have implemented that is not available in the original TILES Card deck. For example, if you have implemented a printer in the firmware and connected it to the TILES Squares, you should create a card for this output primitive.

By following these two steps you ensure that the non-experts only have supported operations available during the ideation phase in the TILES Cards. This way the non-experts are not able to use unsupported interaction primitives, and get confused when it is not supported by the hardware or APIs during the prototyping phase.

B.5 Rule Engine API

1. Introduction

If you are developing your application using the Rule Engine Development Environment, this chapter will explain how you can use the Rule Engine API to

develop your application. This chapter will guide you through the steps of transitioning from your TILES Cards ideation into a working prototype of your application. The steps of this process are listed below.

- a. Map HUMAN ACTIONS cards with TILES API events
- b. Map FEEDBACK cards with TILES API commands
- c. Map SERVICES cards with TILES API sources

After you have finished this Rule Engine API chapter, you should continue with the Test Application step of the Application Development Process to test your application.

The Rule Engine Development Environment

When you have created an application to use the Rule Engine Development Environment, you can navigate to the environment by selecting your application from the list of applications. Your application should look similar to the `garage_control_system` seen below.

At this point, you should have defined the items you want to use in your application by going through the steps of the List Physical Objects step of the Application Development Process.

The screenshot shows the TILES Cloud interface for the 'garage_control_system' application. The top navigation bar includes 'Home', 'Users', 'Applications' (highlighted), 'Primitives', and 'Docs'. Below the navigation bar, the application name 'garage_control_system' is displayed in red, with a 'Start application' button to its right. A user profile for 'Anders' is shown below the application name. The main content area is divided into several sections:

- Development environment:** Shows 'Rule engine' as the active environment, with an 'Edit' button next to it. Below this is the 'IFTTT key'.
- List of items:** Displays 'No item registered for this application.' with a '+ new item' button.
- IFTTT rules:** Displays 'No IFTTT rule registered for this application.' with a '+ new IFTTT rule' button.
- Tile rules:** Displays 'No Tile rule registered for this application.' with a '+ new Tile rule' button.

Figure B-25, Garage Control System application using Rule Engine Dev. Env.

2. Map HUMAN ACTIONS

To define rules for HUMAN ACTIONS cards, you need to find the TILE rule section in your Rule Engine Development Environment, seen in the image below.

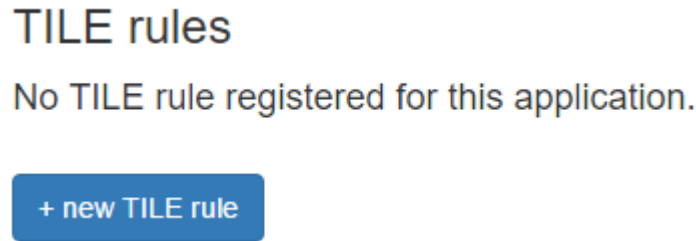


Figure B-26, Add new TILE rule

Click the + **new TILE rule** button to open the TILE rule form seen below.

The image shows the TILE rule form with the following configuration: "IF Double tap ON front_door THEN haptic burst ON front_door". Below the rule configuration are two buttons: "Create" (blue) and "Cancel" (red).

Figure B-27, TILE rule form

In the TILE rule form, the left part is related to the HUMAN ACTIONS cards, while the right part is related to the FEEDBACK cards, as seen in the image below.

The image shows the TILE rule form divided into two sections: "HUMAN ACTION" and "FEEDBACK". The "HUMAN ACTION" section is highlighted with a green border and contains the rule configuration: "IF Double tap ON front_door". The "FEEDBACK" section is highlighted with a yellow border and contains the rule configuration: "THEN haptic burst ON front_door". Below the rule configuration are two buttons: "Create" (blue) and "Cancel" (red).

Figure B-28, TILE rule form divided into HUMAN ACTIONS and FEEDBACK

In the HUMAN ACTIONS section of the TILE rule form, select the relevant input TILES Square from the first dropdown box, and select the trigger action in the second. To conclude the TILE rule, you also need to define a FEEDBACK action. To see how to define the FEEDBACK actions, see the next section.

3. Map FEEDBACK

In the previous section you started creating a TILE rule by defining the input HUMAN ACTION. In this section you will be mapping the FEEDBACK to the second part of the TILE rule form, seen in the image below.

The screenshot shows a form for creating a TILE rule. It is divided into two main sections: **HUMAN ACTION** and **FEEDBACK**. The **HUMAN ACTION** section contains the text "IF Double tap ON front_door THEN". The **FEEDBACK** section contains the text "haptic burst ON front_door". Below the form are two buttons: "Create" (blue) and "Cancel" (red).

Figure B-29, TILE rule form divided into HUMAN ACTIONS and FEEDBACK

In the FEEDBACK section of the TILE rule form, select the desired feedback in the dropboxes before the ON keyword, and select the target TILES Square from the last dropdown box. By clicking Create the rule will be stored in the list as seen below. This specific rule will define that:

IF the front_door TILES Square is Single tapped THEN the garage_door TILES Square will turn LED ON Pink.



IF single tap ON front_door THEN led on FF00FF  ON garage_door 

Figure B-30, Defined TILE rules with HUMAN ACTIONS and FEEDBACK

4. MAP SERVICES

Mapping SERVICES cards into IFTTT rules in the Rule Engine Development Environment can be a bit tricky and it requires some knowledge of external web services. The only available SERVICE in the Rule Engine Development Environment, is the IFTTT rule definition, which allows you to communicate with IFTTT in your application.

Configuring IFTTT

The first thing you need to do is to create an account at IFTTT if you have not already done so. From IFTTT you need to set up the maker channel and write down your personal key. This key must be configured in your TILES application as seen in the image below.

The screenshot shows the configuration screen for the "garage_control_system" application. The user is "Anders". There are two tabs: "Development environment" and "Rule engine". The "Rule engine" tab is active. In the "Rule engine" section, there is a field for "IFTTT key" containing the value "-MXtrSsdb-WQxdmbW-CuA". To the right of the field are two buttons: "Register" (blue) and "Cancel" (orange).

Figure B-31, Add IFTTT personal key to Rule Engine application

Output SERVICE

Now your Rule Engine application is ready to talk to you applets in IFTTT. The next thing you need to do is to create the rule that will trigger the IFTTT applet. Select the **+ new IFTTT rule** button to see the IFTTT rule form below.

Tile IF Double tap ON front_door THEN IFTTT post to IFTTT Trigger name

Create Cancel

Figure B-32, IFTTT rule form1

Select the Tile option in the first dropdown box to define that this rule will be triggered by a HUMAN ACTION. The next two boxes are used to define the input HUMAN ACTIONS as described in 2. Map HUMAN ACTIONS.

In the last input box on the form you should enter the IFTTT trigger name of your applet and click on Create to create the rule. This will add the rule to the list as seen below. This specific rule will define that:

IF the car TILES Square is double tapped THEN the open_garage_door applet in IFTTT is triggered.



Figure B-33, Defined IFTTT rules with HUMAN ACTIONS and SERVICE

Input SERVICE

To define an input SERVICE, meaning that an external SERVICE will trigger a FEEDBACK on your TILES Squares, select the + new IFTTT rule button to see the IFTTT rule form below.

IFTTT IF IFTTT THEN haptic burst ON garage_door

Create Cancel

Figure B-34, IFTTT rule form2

This time select the IFTTT option in the first dropdown box to define that this rule will be triggered by a SERVICE. The remaining form are used to define the FEEDBACK as defined in 3. Map FEEDBACK. When you have defined the FEEDBACK for this rule, hit the Create button, and see your rule appear in the list as seen below. This specific rule will define that:

IF IFTTT applet THEN garage_door TILES Square will vibrate with bursts.



Figure B-35, Defined IFTTT rules with SERVICE and FEEDBACK

The last thing you need to do for this rule to be triggered, is to define your IFTTT applet to send a post request to the URL visible in line with the IFTTT rule as seen on the image above. When a POST request is detected by the TILES Cloud to this specific address, the IFTTT rule will be triggered and the FEEDBACK command will be sent to the defined TILES Square.

5. Example scenario





Description:






1. Single tap on 'Eyewear' will set color on 'Eyewear' to pink.
2. Double tap on 'Eyewear' will vibrate 'Headgear' AND set color on 'Headgear' to the yellow.
3. Tilting 'Eyewear' will turn off LED on 'Headgear'.





Example TILES Cards configuration:

This is just one specific TILES Cards configuration. There are countless ways to explain this scenario with the TILES Cards. You should use the TILES Cards just the way that feels natural to you.

1.

 Tap <small>The user taps the object, either with a single tap or double tap.</small> <small>Human actions A-B</small>	 Eyewear <small>A pair of ordinary glasses or sunglasses.</small> <small>Things T-5</small>	 Eyewear <small>A pair of ordinary glasses or sunglasses.</small> <small>Things T-5</small>	 Color change <small>A light on the object changes from one color to another.</small> <small>Feedback F-6</small>
---	--	--	--
2.

 Tap <small>The user taps the object, either with a single tap or double tap.</small> <small>Human actions A-B</small>	 Eyewear <small>A pair of ordinary glasses or sunglasses.</small> <small>Things T-5</small>	 Headgear <small>A piece of headgear like a cap, a beanie or a helmet.</small> <small>Things T-13</small>	 Color change <small>A light on the object changes from one color to another.</small> <small>Feedback F-6</small>	 Vibrate <small>The object starts vibrating.</small> <small>Feedback F-8</small>
---	--	--	--	---
3.

 Tilt <small>The user tilts the object on one of three axes.</small> <small>Human actions A-6</small>	 Eyewear <small>A pair of ordinary glasses or sunglasses.</small> <small>Things T-5</small>	 Headgear <small>A piece of headgear like a cap, a beanie or a helmet.</small> <small>Things T-13</small>	 Color change <small>A light on the object changes from one color to another.</small> <small>Feedback F-6</small>
--	--	--	--

Rule Engine program example:

example_scenario_1 Start application

Anders

Development environment **Rule engine**

IFTTT key Edit

List of items

Item name	TILE ID	Active	State	Timestamp	
eyewear	unpaired	unpaired	unpaired	unpaired	
headgear	unpaired	unpaired	unpaired	unpaired	

+ new item

TILE rules

IF single tap	ON eyewear	THEN led on FF00FF		ON eyewear		1
IF double tap	ON eyewear	THEN led on FFFF00		ON headgear		2
IF double tap	ON eyewear	THEN haptic burst		ON headgear		2
IF tilt	ON eyewear	THEN led off		ON headgear		3

+ new TILE rule

Figure B-36, Rule Engine example program definition

If you are following the Application Development Process for prototyping your application, please return to step 7 of the process.