SENGAMALA THAYAAR EDUCATIONAL TRUST WOMEN'S COLLEGE

(Affiliated to Bharathidasan University, Tiruchirappalli)
Recognized by UGC under section 2(f) & 12(B)
(Accredited with 'A' Grade {CGPA: 3.45/4.00} by NAAC)
(An ISO 9001: 2015 Certified Institution)

SILVER JUBILEE YEAR (2019-2020)

COURSE MATERIAL

COURSE : M.Sc[IT]

TITLE OF THE PAPER : INTERNET OF THINGS

PAPER CODE : P16IT41

Prepared By,

Mrs.K.SAVIMA,

ASSISTANT PROFESSOR,

S.T.E.T.WOMEN'S COLLEGE,
MANNARGUDI.

STET WOMEN'S COLLEGE

Unit – V

Creating the IoT projects:

Sensor project

A **sensor** is a device that detects and responds to some type of input from the physical environment. ... The output is generally a signal that is converted to human-readable display at the **sensor** location or transmitted electronically over a network for reading or further processing.

Put simply, a sensor converts stimuli such as heat, light, sound and motion into electrical signals. These signals are passed through an interface that converts them into a binary code and passes this on to a computer to be processed.

Sensors enhance and replace various human senses in automated assembly equipment. They respond to light, electromagnetic fields, sound or pressure. ... An analog **sensor** is wired into a circuit so that it will have an output that falls within a certain range. Then, the value can assume any possible value within that range.

All **types of sensors** can be basically classified into analog **sensors** and digital **sensors**. But, there are a few **types of sensors** such as temperature **sensors**, IR **sensors**, ultrasonic **sensors**, pressure **sensors**, proximity **sensors**, and touch **sensors** are frequently used in most of the electronics applications.

A **good sensor** obeys the following rules: it is sensitive to the measured property. it is insensitive to any other property likely to be encountered in its application, and. it does not influence the measured property.

a **sensor** is a device, module, machine, or subsystem whose purpose is to detect events or changes in its environment and send the information to other electronics, frequently a computer processor. A sensor is always used with other electronics.

Sensors are used in everyday objects such as touch-sensitive elevator buttons (tactile sensor) and lamps which dim or brighten by touching the base, besides innumerable applications of which most people are never aware. With advances in micromachinery and easy-to-use microcontroller platforms, the uses of sensors have expanded beyond the traditional fields of temperature, pressure or flow measurement, for example into MARG

ST. JOSEPH'S COLLEGE

sensors. Moreover, analog sensors such as potentiometers and force-sensing resistors are still widely used. Applications include manufacturing and machinery, airplanes and aerospace, cars, medicine, robotics and many other aspects of our day-to-day life. There are a wide range of other sensors, measuring chemical & physical properties of materials. A few examples include optical sensors for Refractive index measurement, vibrational sensors for fluid viscosity measurement and electro-chemical sensor for monitoring pH of fluids.

A sensor's sensitivity indicates how much the sensor's output changes when the input quantity being measured changes. For instance, if the mercury in a thermometer moves 1 cm when the temperature changes by 1 °C, the sensitivity is 1 cm/°C (it is basically the slope dy/dx assuming a linear characteristic). Some sensors can also affect what they measure; for instance, a room temperature thermometer inserted into a hot cup of liquid cools the liquid while the liquid heats the thermometer. Sensors are usually designed to have a small effect on what is measured; making the sensor smaller often improves this and may introduce other advantages.

Technological progress allows more and more sensors to be manufactured on a microscopic scale as microsensors using MEMS technology. In most cases, a microsensor reaches a significantly faster measurement time and higher sensitivity compared with macroscopic approaches. Due the increasing demand for rapid, affordable and reliable information in today's world, disposable sensors—low-cost and easy-to-use devices for short-term monitoring or single-shot measurements—have recently gained growing importance. Using this class of sensors, critical analytical information can be obtained by anyone, anywhere and at any time, without the need for recalibration and worrying about contamination.

The following projects are based on sensors. This list shows the latest innovative projects which can be built by students to develop hands-on experience in areas related to/ using sensors.

1. IoT using Raspberry Pi

2. Automatic Solar Tracker

3. 7 Robots (Combo Course)

4. Gesture Controlled Robot

5. Gesture Based Robotics

6. IoT using Arduino

STELLA / GE

7. Sensor Guided Robotics

8. Automated Street Lighting

9. Smart Irrigation System

10. Smart Water Monitoring

11. Smart Building using IoT

12. 5 IoT Projects (Combo Course)

13. 4 Smart Energy Projects

14. 2 IoT Projects (Combo Course)

15. Animatronic Hand

16. Swarm Robotics

17. Automated Railway Crossing

18. Maze Solver Robot

19. Edge Detection Robot

20. 4 Mechatronics Projects (Combo Course)

STET WOMEN'S COLLEGE

Actuator project

An **actuator** is a component of a machine that is responsible for moving and controlling a mechanism or system, for example by opening a valve. In simple terms, it is a "mover". ... When it receives a control signal, an **actuator** responds by converting the signal's energy into mechanical motion.

A valve **actuator** is a mechanical device that uses a power source to operate a valve. This power source can be electric, pneumatic (compressed air), or hydraulic (the flow of oil). There

are two main **types** of **actuators**, one for each of the two main **types** of valves that require them. They are rotary and linear.

There are four common types of actuators: manual, pneumatic, hydraulic, and electric.

What is actuator in HVAC?

Dampers are the final control devices for almost all airflow in **HVAC** systems. **Actuators** are the interface between the control system and the mechanical system and are critical to accurate control. Typically, 80% or more of direct digital control (DDC) outputs in the **HVAC** portion of the system go to **actuators**.

What is an actuator example?

In a mechanical device, an **actuator** is a component that turns the control signal into movement. **Examples** of **actuators** include: Electric motors. Solenoids. Hard drive stepper motors.

Where is actuator used?

Electric rotary **actuators** are **used** primarily in automation applications when a gate, valve, etc. requires controlled movement to particular rotational positions. They are **used** in a wide range of industries where positioning is needed. The **actuators** are driven by various motor types, voice coils, etc.

What are the three types of actuators?

It describes different types of Actuators viz. Hydraulic actuator, Pneumatic actuator, Electrical actuator, **Mechanical** actuator, Thermal or Magnetic actuator, Soft actuator etc.

What is the use of actuator?

An **actuator** is something that converts energy into motion. It also can be used to apply a force. An **actuator** typically is a mechanical device that takes energy — usually energy that is created by air, electricity or liquid — and converts it into some kind of motion.

What is actuator system?

Actuation systems are the elements of control **systems** which are responsible for transforming the output of a microprocessor or control **system** into a controlling action on a machine or device.

How does the actuator work?

An **actuator** is a motor that converts energy into torque which then moves or controls a mechanism or a system into which it has been incorporated. It can introduce motion as well as

prevent it. An **actuator** typically runs on electric or pressure (such as hydraulic or pneumatic). ...

How does a linear actuator work?

An electric **linear actuator** is a device that converts the rotational motion of a AC or DC motor into **linear** motion – that is, it will provide both push and pull movements. By pushing and pulling it is possible to lift, drop, slide, adjust, tilt, push or pull objects, simply by pushing a button.

Controller

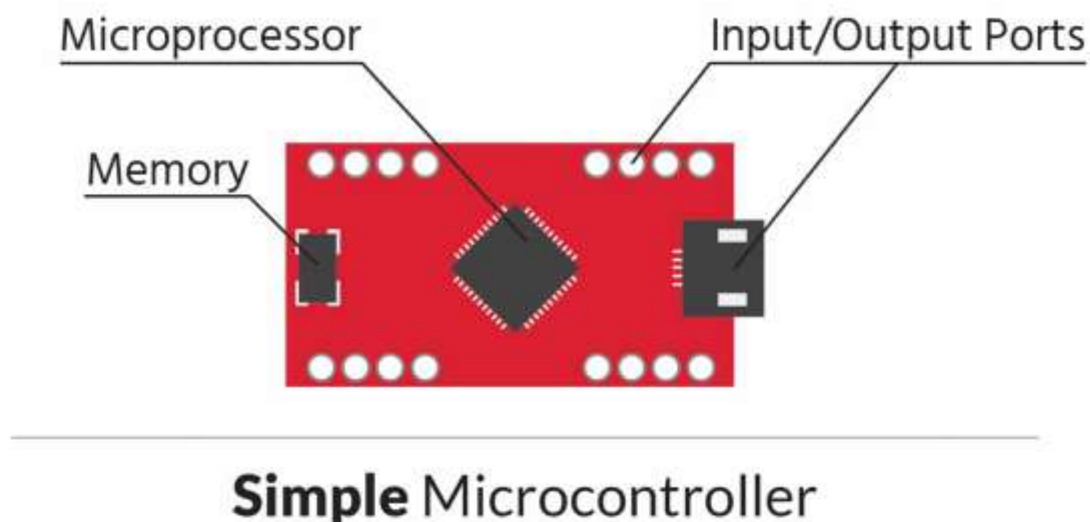
What is a controller in IoT?

IoT Controllers. Qorvo's innovative ultra-low-power wireless data communication **controller** chips enable smart home applications and the Internet of Things (**IoT**). Our radio communication chips are optimized for remote controls, smart home gateways, and end devices, each with their specific features and needs.

What is the role of controller service in IoT?

It handles all the number crunching and local data manipulation and decision-making. ... The input ports collect data from sensors. While the outputs support any necessary actuation or local control in the **IoT** device. Usually, microcontrollers control various devices or subsystems within embedded applications.

The Control System | The Brain in an IoT Device



We recently published an introduction to IoT Sensors and Actuators, where we learned that the enabling technology is a transducer, a physical device that converts one form of energy into another. Now let's look at how this energy conversion is used as input for a control system in an IoT device.

The Control System

Your IoT device may be smaller than a coin or larger than a refrigerator. It may perform a simple sensing function and send raw data back to a control center. It may combine data from many sensors, perform local data analysis, and then take action. Additionally, your device could be remote and standalone or be co-located within a larger system.

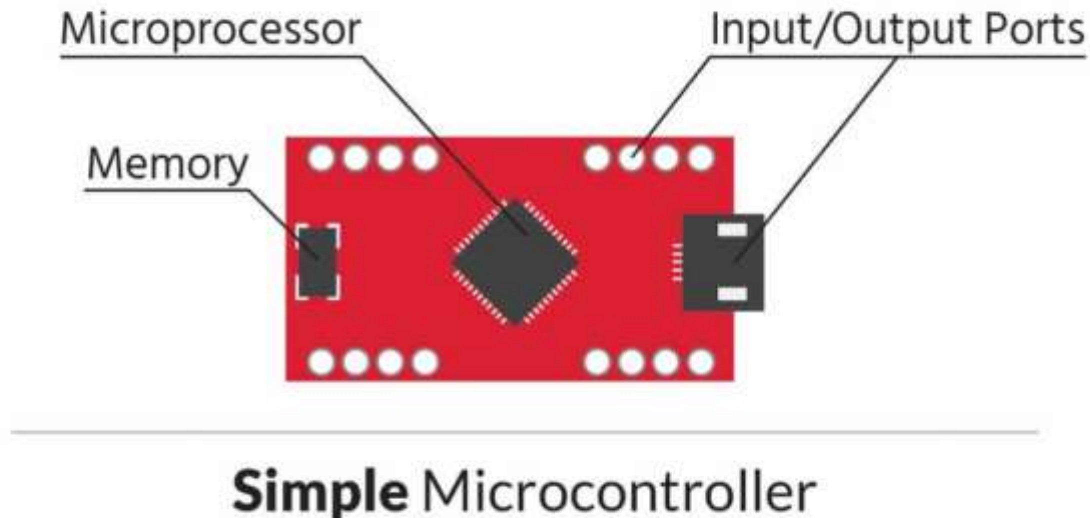
Regardless of the function, environment, or location, your IoT device requires two components, a brain and connectivity. The "brain" provides local control (or decision-making). Your device's function will determine the size and capabilities of the brain component. Connectivity is needed to communicate with external control. The environment and location of your device will determine how it connects.

The "Brain" of the IoT Device

Your IoT device will most likely use a microcontroller as its brain. Think of a microcontroller as a small computer with a microprocessor core, memory, and input/output (I/O) ports. The microprocessor core of your microcontroller is a central processing unit. It handles all the number crunching and local data manipulation and decision-making. The memory includes Read Only Memory (ROM) and Random Access Memory (RAM). ROM stores the microcontroller's software program. RAM stores and receives data while also supporting number crunching. The final microcontroller component, the I/O ports, may be either digital or analog. The input ports collect data from sensors. While the outputs support any necessary actuation or local control in the IoT device.

Usually, microcontrollers control various devices or subsystems within embedded applications. By integrating the microprocessor, memory, and input/outputs, microcontrollers reduce cost and make development easier. This makes it more affordable and less complicated to control many IoT devices.

STET WOMEN'S COLLEGE



Simple **Microcontroller**

To determine which microcontroller to use for an IoT application, you first need to understand a few basic specifications:

- *Microprocessor Type ("Brain")*

How fast your brain needs to think (clock speed) and how much information it can handle (data I/O bus size) determines the microprocessor type. Depending on your application, you may have a very simple microprocessor or a much faster, bigger, power hungry one. While you don't need to be an expert on microprocessors, you should have an idea of what you want it to be able to do.

- *Amount of Memory*

ROM "stores" your application program in the microcontroller. The size of the program is a function of the complexity. RAM performs two functions: It reads and writes data for file storage, and it holds data waiting for processing by the microprocessor. An important distinction between ROM and RAM is volatility. ROM is non-volatile, meaning that the data remains stored with or without power. RAM is volatile, meaning it may lose data if it is without power. This is an important factor for IoT devices which may be prone to an interruption of the power supply.

STET WOMEN'S COLLEGE

- *Operating Voltage and Current*

  The device needs adequate voltage to support its sensors and power sources. If a device is part of a larger system, it may use a local output voltage. Generally, sensor and battery technologies operate within a compatible voltage range. You will need higher output currents if you need control or activation (such as operating a motor).

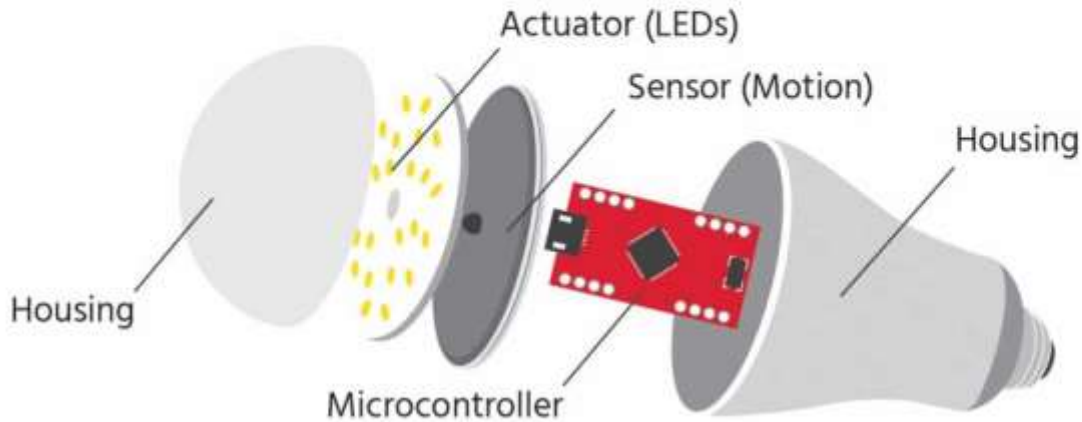
- *Number and Type of Input/output (I/O) Ports*

  The I/O ports are the connection to your device sensors and actuators. There may be few, or many, depending on the application. The ports may act as inputs from sensor to device, as outputs from device to sensor (or actuator), or both. As mentioned above, I/O ports may be digital or analog. Digital ports are for simple logic, or yes/no type input. Examples include, closing a switch, tripping a wire, presenting/not presenting something. When acting as an output, a digital port may turn something on or off. Analog ports are for continuous input/output like a temperature or speed. Keep in mind; microcontrollers are digital devices so signals moving in and out must also be in digital format. Inputs from analog sensors use an analog-to-digital converter (ADC). The purpose of the ADC is to convert data into a format that is usable by the microcontroller. Once you define your IoT device application, you can determine the type and number of I/O ports you need.

- *Control Interface*

  A control interface is a protocol allowing peripheral devices and the microcontroller to communicate with one another. Depending on the application, you may need a specific control interface. There are different ways the interface may facilitate communication. It may use an inter-integrated circuit (I2C), a serial peripheral interface (SPI), or a controller area network (CAN) communications protocol. While you do not need to know technical details about these, you do need to know which protocols your system requires. Those will determine if you need a control interface and how that interface should communicate.

STET WOMEN'S COLLEGE



Camera

Cameras are expected to become key sensor devices for various internet of things (**IoT**) applications. ... Our approach towards data security for smart **cameras** is rooted on protecting the captured images by signcryption based on elliptic curve cryptography (ECC).

When does a camera become a sensor?

"What most still refer to as 'CCTV' cameras have come a long way since the grainy days of analogue. Today's cameras are mostly digital, very high resolution and now extend beyond just capturing video — to analysing and providing insight into what's happening on-screen," said Crosbie.

"Physical retailers are often at a disadvantage to e-commerce sites because they lack tangible data about store activity such as: how many people visit their shops at different times of day and week; where people move throughout shops (pathmaps); where people linger longest (heatmaps), and which items they handle most – all of which Prism can provide.

One of the most profitable discoveries a retailer can make is which products are getting handled the most, but *not* purchased. Knowing the underlying reasons – problems with quality, sizing, pricing – a store manager can course correct in time. I'm talking to a large petrol station retailer that has a simple goal of converting £1 of this per customer into sales, which it reckons would

seriously boost the bottom line. This is the kind of problem that's easy to solve with IoT and data analytics technology.

Of course, when retailers improve the store to sell more, the in-store experience always gets better. Popular items that customers want are easier to find, queues don't build up, and promotions are targeted better. These don't just directly benefit customers, but they improve staff morale as well, which indirectly benefits customers and the overall brand."

Using an IoT service platform

An IoT platform plays a pivotal role for smart device vendors and startups, who can use it to equip their products with remote control and real-time monitoring functions, configurable alerts and notifications, pluggable **cloud** services, and integration with consumers' smartphones and other devices.

What are different IoT platforms?
IoT Platform Comparison

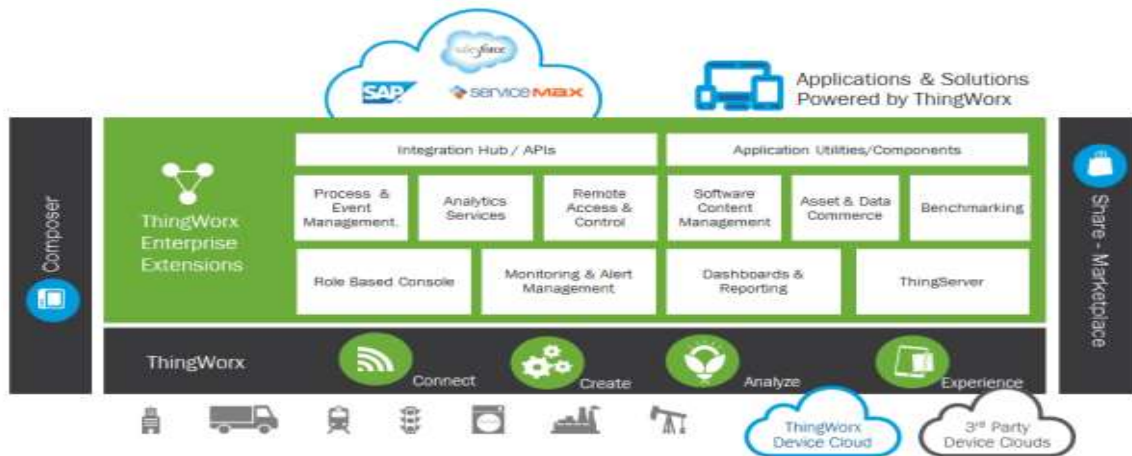
IoT platform	Services
Particle	Hardware, Connectivity, Device Cloud, and Apps.
Salesforce IoT Cloud	Data from customers, partners, devices, and sensors.
ThingWorx	End-to-end Industrial IoT platform.
IBM Watson IoT	Connection Service, Analytics Service, Blockchain Service.

Top 11 Cloud Platforms for Internet of Things (IoT)

1. Thingworx 8 IoT Platform

Thingworx is one of the leading IoT platforms for industrial companies, which provides easy connectivity for devices. It enables the experience from today's connected world. Thingworx 8 is a better, faster, easier platform, providing the functionality to build, deploy, and extend industrial projects and apps.

STET WOMEN'S COLLEGE



Thingworx is an IoT platform designed by PTC for development of enterprise app development. It offers basic features, such as:

- Easy connectivity with electronic devices, like sensors and RFIDs
- You can work remotely once you are done with the setup
- Pre-built widgets for the dashboard
- Remove Complexity of the project
- Integrated machine learning

Pros

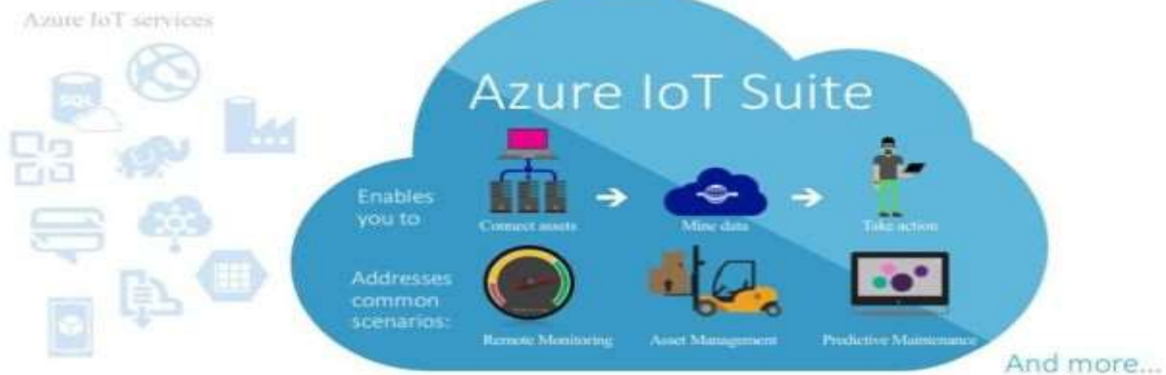
- Easy web page designs for customers
- Easy to manage devices
- Simple connectivity solutions

Cons

- Difficult to use with custom programs in C#
- Hard to manage complex systems.
- The limitation to install edge program on a custom platform.

2. Microsoft Azure IoT Suite

Microsoft Azure provides multiple services to create IoT solutions. It enhances your profitability and productivity with pre-built connected solutions. It analyzes untapped data to transform business. This provides the solutions for a small PoC to Rolling out your ideas. Azure Suite can easily analyze and act on new data.



Azure IoT Suite provides features like:

- Easy Device Registry.
- Rich Integration with SAP, Salesforce, Oracle, WebSphere, etc
- Dashboards and visualization
- Real-time streaming

Pros

- Offers third-party services
- Secure and scalable
- High availability

Cons

- Requires management
- Expensive
- No support for bugs

3. Google Cloud's IoT Platform

Google's platform is among the best platforms we currently have. Google has an end-to-end platform for Internet-of-Things solutions. It allows you to easily connect, store, and manage IoT data. This platform helps you to scale your business.

Their main focus is on making things easy and fast. Pricing on Google Cloud is done on a per-minute basis, which is cheaper than other platforms.

Google Cloud's IoT platform provides features, including:

- Provides huge storage
- Cuts cost for server maintenance
- Business through a fully protected, intelligent, and responsive IoT data
- Efficient and scalable
- Analyze big data

Pros

- Fastest input/output
- Lesser access time
- Provides integration with other Google services

Cons

- Most of the components are Google technologies
- Limited programming language choices

4. IBM Watson IoT Platform

IBM Watson is a powerful platform backed by IBM 's the Bluemix and hybrid cloud PaaS (platform as a service) development platform. By providing easy sample apps and interfaces for IoT services, they make it accessible to beginners. You can easily try out their sample to see how it works, which makes it stand out from other platforms.

Users can get the following features:

- Real-time data exchange
- Secure Communication
- Cognitive systems
- Recently added data sensor and weather data service

Pros

- Process untapped data
- Handle huge quantities of data
- Improve customer services

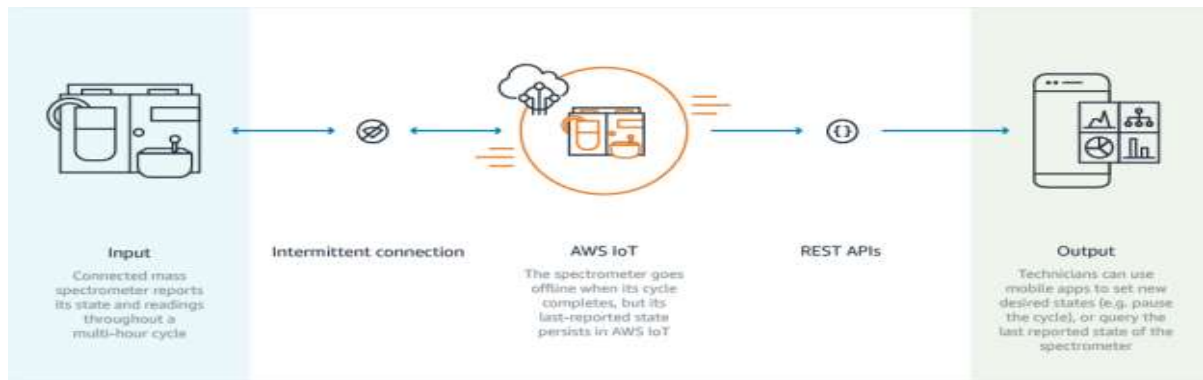
Cons

- Need a lot of maintenance.
- Take time for Watson integration
- High switching cost.

5. AWS IoT Platform

Amazon made it much easier for developers to collect data from sensors and Internet-connected devices. They help you collect and send data to the cloud and analyze that information to provide the ability to manage devices.

You can easily interact with your application with the devices even they are offline.



Main features of the AWS IoT platform are:

- Device management
- Secure gateway for devices
- Authentication and encryption
- Device shadow

Pros

- Good integration with laas offering.
- Price dropped over the last six years
- Open and flexible

Cons

- A big learning curve for AWS
- Three outages in the last 2 years
- Not secure for hosting critical enterprise applications

6. Cisco IoT Cloud Connect

Cisco Internet of Things accelerates digital transformation and actions from your data. Cisco IoT Cloud Connect is a mobile, cloud-based suite. It offers solutions for mobile operators to provide phenomenal IoT experience. It provides flexible deployment options for your devices.

The main feature of the Cisco Cloud Connect:

- Data and voice connectivity
- Device and IP session report
- Billing is customizable
- Flexible deployment options

7. Salesforce IoT Cloud

Salesforce IoT Cloud is powered by Salesforce Thunder. It gathers data from devices, websites, applications, and partners to trigger actions for real-time responses. Salesforce combined with IoT delivers improved customer service.

STE

Key features of Salesforce IoT Cloud:

- Enhanced data collection
- Improved customer engagement
- Real-time event processing
- Technology optimization

Pros

- Scale to billions of devices and messages.
- Easy UI designs to connect with customers.

Cons

- Security liability
- Flexibility limitations

8. Kaa IoT Platform

Kaa is an open-source, multipurpose, middleware platform for complete end-to-end IoT development and smart devices. It reduces cost, risk, and market time. Also, Kaa offers a range of IoT tools that can be easily plugged in and implemented in IoT use cases.

It provides many features that make it unique, such as:

- Reduce development time
- Open source and free
- Easy and direct device implementation
- Reduce marketing time
- Handle millions of devices

Pros

- Ease of use
- Third-party integration
- Data security

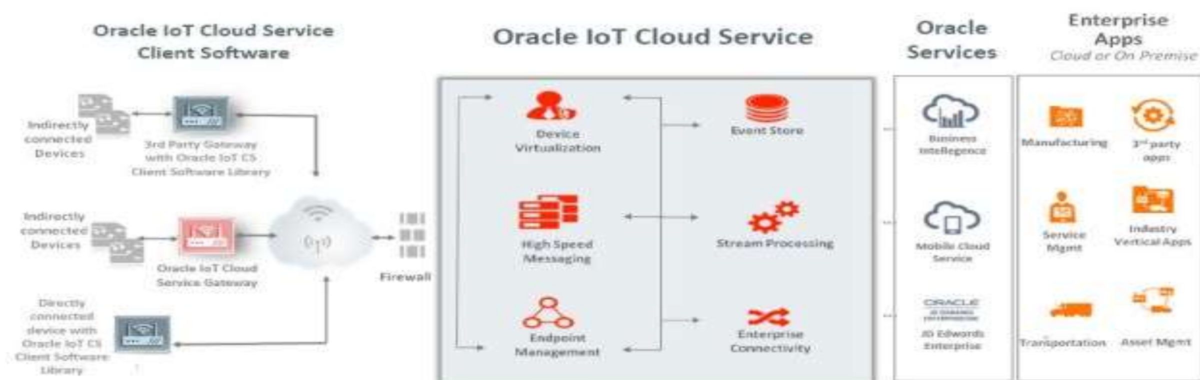
Cons

- Not able to deploy applications based on the PaaS model

9. Oracle IoT Platform

Oracle offers real-time Internet of Things data analysis, endpoint management, and high speed messaging where the user can get real-time notification directly on their devices. Oracle IoT

cloud service is a Platform as a Service (PaaS), cloud-based offering that helps you to make critical business decisions.



Features offering to users

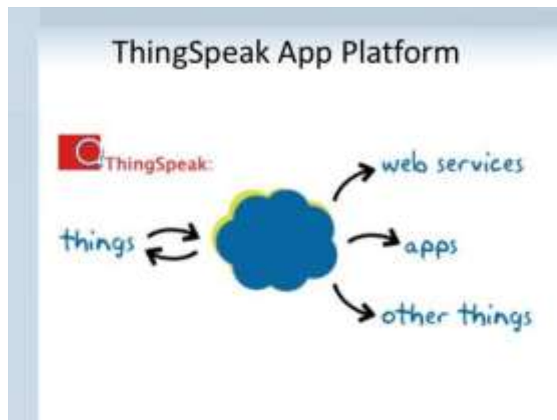
- Secure and scalable
- Real-time insight
- Integrated
- Faster to market

Pros

- Device visualization
- High speed messaging
- Event store

10. Thingspeak IoT Platform

Thingspeak is an open-source platform that allows you to collect and store sensor data to the cloud. It provides you the app to analyze and visualize your data in Matlab. You can use Arduino, Raspberry Pi, and Beaglebone to send sensor data. You can create a separate channel to store data.



Features of Thingspeak:

- Collect data in private channels
- App integration
- Event scheduling
- MATLAB analytics and visualization

Pros

- Free hosting for channels
- Easy visualization
- Provides additional features for Ruby, Node.js, and Python

Cons

- Limited data uploading for API
- ThingSpeak API can be a hurdle for beginners

11. GE Predix IoT Platform

Predix is the world's first industrial platform. Predix was designed to target factories and provides simple ecosystem. It can directly analyze data from the machine and store. GE wants to provide the growing industrial Internet of Things for its cloud platform. This platform is secure and scalable.

According to the customers their IoT platform can:

- Optimize assets and operations
- Provides key performance data
- Reduces unplanned downtime
- Real-time operational data

Selecting an IoT

The Internet of Things (IoT) brings plenty of opportunities for System Integrators (SI). The topic is hot, and customers are keen to harvest value from smart connected devices, digital twins, and IoT-enabled business processes. It's also true that there's no one de-facto "IoT-in-a-box" solution today, and integrators deliver most implementations using a variety of technologies.

The first dilemma for SI is a choice between building the platform that "ticks all technical requirement boxes" and finding a suitable platform on the market. Platform development is time-consuming, and not alwats rewarding; rivals can take the market share while the SI develops the "ideal platform."

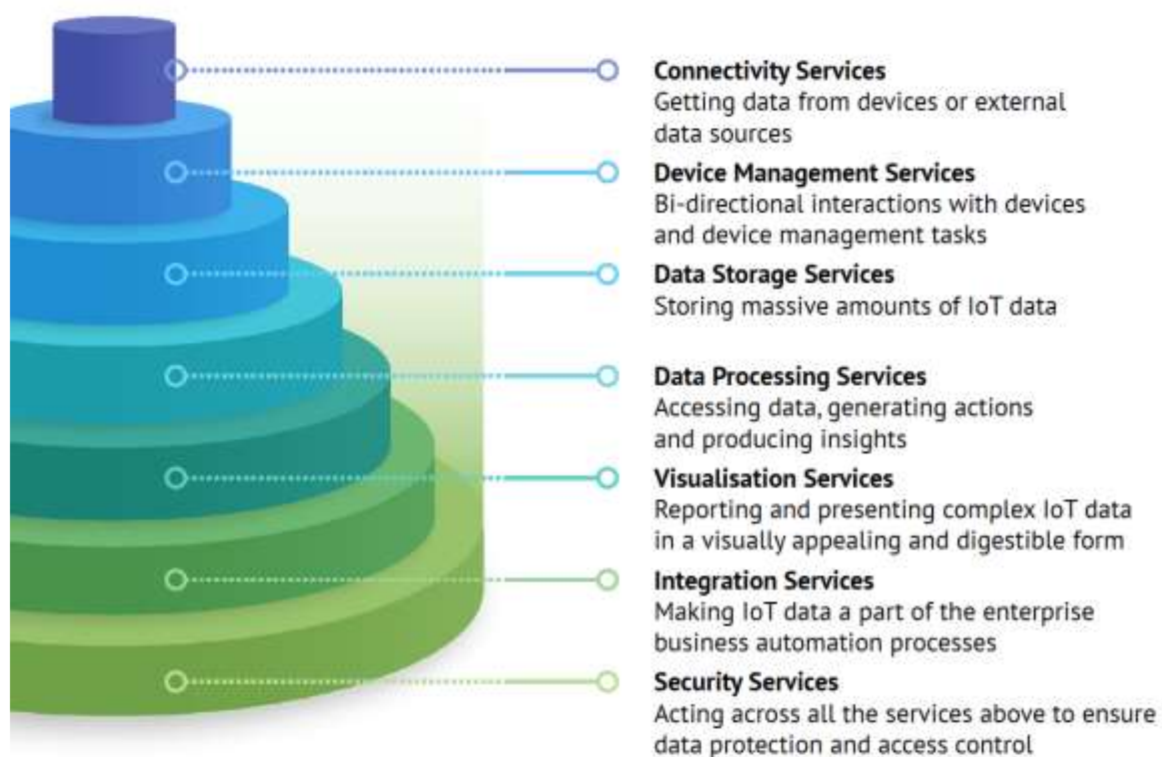
One should not forget about everlasting platform maintenance costs, including new industrial protocol support and certification and new regulatory requirements support.

Those who want to choose the right IoT platform need to consider criteria that also keep core SI business principles in mind:

- Complete profitable projects and deliver on time
- Try to win more projects with the same customer after the first successful implementation
- Reuse skills and resources where possible

A good IoT platform helps the System Integrator to achieve all above, but the SI needs to make a tough choice among several hundreds of self-described IoT platforms available on the market.

Before moving to selection criteria, let's define the IoT platform and its services.



Despite some "platforms" can formally claim all services available, it does not make them the right choice for the integrator.

There are three areas SI needs to evaluate the IoT platform from a technical, business, and partnership perspective.

Technical Evaluation

The more obvious selection criteria, including connectivity, device management, data storage, data processing, visualization, integration, and security, are covered in the full version of this article.

There are also less apparent criteria that integrators often overlook during the IoT platform selection process.

Platform Development and Diagnostic Tools

The IoT platform itself is the development tool, and its administrative module is an Integrated Developer Environment (IDE). Therefore, it should allow not only application logic development but also provide a full set of debugging capabilities.

Scalability

First, the platform should allow building a horizontal cluster with workload balancing capabilities.

Second, it's multi-layer architecture support. Each layer, including edge servers, data collection, data storage, data processing and analytics, application servers and UI servers, should scale independently.

Performance

Too many processing cores or too much memory required for data collection and manipulation can make IoT solution unreasonably expensive. The platform should allow building cost-effective distributed architectures keeping the solution performance high.

DevOps

The platform should allow fast code transfer between development, testing, production environments and provide integration with GitHub/SVN.

On-Premise/Cloud and Edge Code Base Compatibility

Unfortunately, it's not a rare case when the code for edge computing differs from the platform code. It often happens when the IoT platform vendor tries to catch up with the market trend towards edge computing by acquisitions.

The same set of development tools and code portability between edge and on-premise/cloud environments is a big time saver for the integrator.

Business Evaluation

- References and experience: A universal IoT platform means a mix of industries and use cases in the list of references.
- IoT platform focus: Ideally, the platform should be the core business for the vendor. But this is not always the case.
- Roadmap: Is it public or available upon request?
- Platform deployment options should include on-premise and be public Cloud provider agnostic.

Partnership Program

- Education roadmap: The education program from the platform should cover all IoT solution roles: administrators, implementation engineers, developers, UI developers, data scientists, architects, etc.
- Documentation: The same as for education purposes, the Platform documentation should be role-based, detailed, and up-to-date.
- Pricing policy: The pricing policy should be understandable. For long-term strategic relationships, it’s also important to be clear about annual price harmonizations. Transparent pricing allows budget projects accurately.

Final Thoughts

The IoT can be the next Big Thing for many System Integrators, and the right platform choice determines long-term success.

While an idea to build a simple IoT platform can be a reasonable option for an enterprise with only one IoT application, it’s not usually the case for a System Integrator. Most of SIs prefer to partner with a specific IoT platform.

Platform

What is an IoT platform?

An IoT platform is a multi-layer technology that enables straightforward provisioning, management, and automation of connected devices within the Internet of Things universe. It basically connects your hardware, however diverse, to the cloud by using flexible connectivity options, enterprise-grade security mechanisms, and broad data processing powers. For developers, an IoT platform provides a set of ready-to-use features that greatly speed up development of applications for connected devices as well as take care of scalability and cross-device compatibility.

Thus, an IoT platform can be wearing different hats depending on how you look at it. It is commonly referred to as middleware when we talk about how it connects remote devices to user applications (or other devices) and manages all the interactions between the hardware and the application layers. It is also known as a cloud enablement platform or IoT enablement platform to pinpoint its major business value, that is empowering standard devices with cloud-based applications and services. Finally, under the name of the IoT application enablement platform, it shifts the focus to being a key tool for IoT developers.

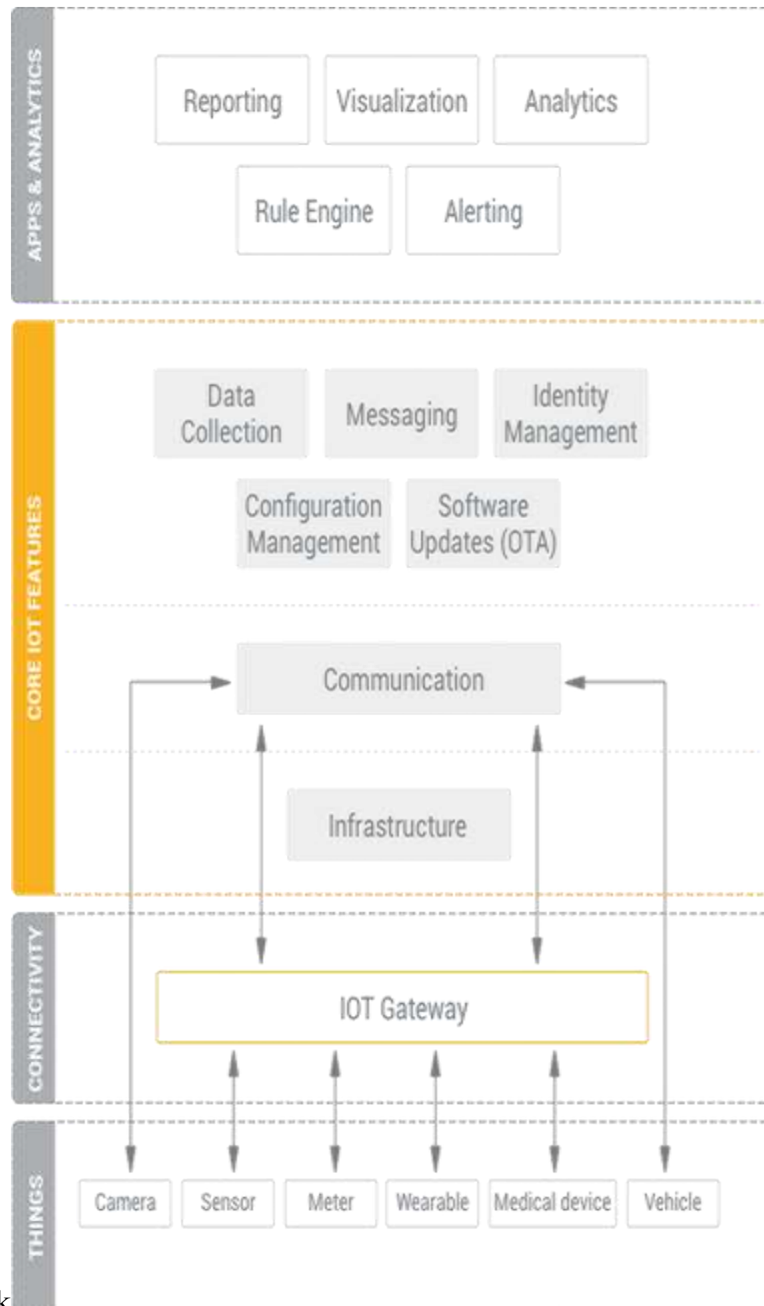
IoT platform as the middleware

IoT platforms originated in the form of IoT middleware, which purpose was to function as a mediator between the hardware and application layers. Its primary tasks included data collection from the devices over different protocols and network topologies, remote device configuration and control, device management, and over-the-air firmware updates.

To be used in real-life heterogeneous IoT ecosystems, IoT middleware is expected to support integration with almost any connected device and blend in with third-party applications used by the device. This independence from underlying hardware and overhanging software allows a single IoT platform to manage any kind of connected device in the same straightforward way.

Modern IoT platforms go further and introduce a variety of valuable features into the hardware and application layers as well. They provide components for frontend and analytics, on-device data processing, and cloud-based deployment. Some of them can handle end-to-end IoT solution implementation from the ground up.

STET WOMEN'S COLLEGE



IoT platform technology stack

In the four typical layers of the IoT stack, which are things, connectivity, core IoT features, and applications & analytics, a top-of-the-range IoT platform should provide you with the majority of IoT functionality needed for developing your connected devices and smart things. Your devices connect to the platform, which sits in the cloud or in your on-premises data center, either directly or by using an IoT gateway. A gateway comes useful whenever your endpoints aren't capable of direct cloud communication or, for example, you need some computing power on edge. You can also use an IoT gateway to convert protocols, for example, when your

endpoints are in LoRaWan network but you need them to communicate with the cloud over MQTT.

An IoT platform itself can be decomposed into several layers. At the bottom there is the infrastructure level, which is something that enables the functioning of the platform. You can find here components for container management, internal platform messaging, orchestration of IoT solution clusters, and others.

The communication layer enables messaging for the devices; in other words, this is where devices connect to the cloud to perform different operations.

The following layer represents core IoT features provided by the platform. Among the essential ones are data collection, device management, configuration management, messaging, and OTA software updates.

Sitting on top of core IoT features, there is another layer, which is less related to data exchange between devices but rather to processing of this data in the platform. There is reporting, which allows you to generate custom reports. There is visualization for data representation in user applications. Then, there are a rule engine, analytics, and alerting for notifying you about any anomalies detected in your IoT solution.

Importantly, the best IoT platforms allow you to add your own industry-specific components and third-party applications. Without such flexibility adapting an IoT platform for a particular business scenario could bear significant extra cost and delay the solution delivery indefinitely.

Advanced IoT platforms

There are some other important criteria that differentiate IoT platforms between each other, such as scalability, customizability, ease of use, code control, integration with 3rd party software, deployment options, and the data security level.

- **Scalable (cloud native)** – advanced IoT platforms ensure elastic scalability across any number of endpoints that the client may require. This capability is taken for granted for public cloud deployments but it should be specifically put to the test in case of an on-premises deployment, including the platform's load balancing capabilities for maximized performance of the server cluster.
- **Customizable** – a crucial factor for the speed of delivery. It closely relates to flexibility of integration APIs, louse coupling of the platform's components, and source code transparency. For small-scale, undemanding IoT solutions good APIs may be enough to fly, while feature-rich, rapidly evolving IoT ecosystems usually require developers to have a greater degree of control over the entire system, its source code, integration

interfaces, deployment options, data schemas, connectivity and security mechanisms, etc.

- **Secure** – data security involves encryption, comprehensive identity management, and flexible deployment. End-to-end data flow encryption, including data at rest, device authentication, user access rights management, and private cloud infrastructure for sensitive data – this is the basics of how to avoid potentially compromising breaches in your IoT solution.

Cutting across these aspects, there are two different paradigms of IoT solution cluster deployment offered by IoT platform providers: a public cloud IoT PaaS and a self-hosted private IoT cloud.

IoT cloud enablement

An IoT cloud is a pinnacle of the IoT platforms evolution. Sometimes these two terms are used interchangeably, in which case the system at hand is typically an IoT platform-as-a-service (PaaS). This type of solution allows you to rent cloud infrastructure and an IoT platform all from a single technology provider. Also, there might be ready-to-use IoT solutions (IoT cloud services) offered by the provider, built and hosted on its infrastructure.

However, one important capability of a modern IoT platform consists in a private IoT cloud enablement. As opposed to public PaaS solutions located at a provider's cloud, a private IoT cloud can be hosted on any cloud infrastructure, including a private data center. This type of deployment offers much greater control over the new features development, customization, and third-party integrations. It is also advocated for stringent data security and performance requirements.

What your business can do with an IoT platform

An IoT platform plays a pivotal role for **smart device vendors and startups**, who can use it to equip their products with remote control and real-time monitoring functions, configurable

alerts and notifications, pluggable cloud services, and integration with consumers' smartphones and other devices.

Another broad application of the IoT platform is cost optimization for **companies in the industrial, agriculture, and transportation sectors** through remote monitoring of devices and vehicles, predictive maintenance of equipment, collecting sensor data for real-time production analytics and ensuring safety, and end-to-end cargo delivery tracking.

Large-scale IoT clouds are typical solutions for **CSPs, smart city and smart energy integrators**. By using an IoT platform, these companies develop IoT infrastructures for delivering all kinds of new services for regular customers, public service companies, and giant corporations. Among them are connected car services, smart grid metering, city-wide air quality monitoring, smart building deployments, and numerous others.

Finally, an IoT platform is the essential technology for improving customer experience in **retail, healthcare, hospitality, and travelling domains**. It is used to enable highly personalized services and ensure stress-free

interaction between the customer and the company. A case in point is remote patient monitoring and treatment solutions, which are incredibly convenient to use and save a person much time on regular visits to the hospital. Collecting thorough patient data becomes effortless with the IoT, whereas retailers and hospitality companies use rich data collection to create personal offerings and run effective marketing.

Strategic trends for IoT platforms

As Gartner pointed out in their technology forecast for 2018, the next generation of business ecosystems will be increasingly more digital, intelligent, and connected. The IoT plays a pivotal role in what they call "an intelligent digital mesh", thus setting a higher bar for modern IoT platforms in several areas.

- **Digital twins** – offer a more powerful way to monitor, control, and manage assets. Digital twins provide a comprehensive digital representation of real-world devices and systems, thus improving their state monitoring and enabling faster responses to external and internal events. Implementation of digital twins require IoT platforms to provide a highly flexible device management capabilities, which could cope with any level of sophistication depending on the use case. The applications of digital twins vary widely from asset inventory and predictive maintenance to event simulation and usage analytics. In the nearest future they are expected to become a keystone of every efficient IoT ecosystem.
- **Intelligent things** – utilize AI and machine learning to extract more insights from collected data and optimize their interactions within an IoT ecosystem. As basic IoT use cases have been successfully gaining ground, further enhancement offered by AI will galvanize even greater progress. Autonomous vehicles and robots at manufacturing facilities maximize production and delivery speed while unparalleled surveillance capacity provided by AI-supported security cameras radically transforms criminal-justice systems. The examples are many in every sector of economy. To allow for these cutting-edge solutions, IoT platforms must be designed to support flexible integration with AI systems and offer scalable, resilient device orchestration.
- **Cloud to the edge** – places computing and data processing powers closer to managed entities in an IoT ecosystem. The combination of cloud and edge computing architectures enables all the benefits of a flexible cloud-native model, where separate services can be managed and distributed across connected assets in a scalable manner, while at the same time ensure effective operation of disconnected entities and allow

them take faster actions in response to new data. Considering other related tasks in regard to cluster management in private cloud and multicloud environments, it is clear that an IoT platform have grown out of being a mere development tool and should be capable of handling DevOps as effectively.

Connecting the dots

IoT solutions are getting inevitably more complex and dynamic. They involve larger ecosystems of devices and evolve much faster than traditional enterprise application software. With the proliferation of all types of remote interactions between devices and humans, IoT solutions are also spearheading a new paradigm for customer-oriented digital experience. Their complexity may seem intimidating at first but, in fact, taking advantage of the IoT is feasible at a fraction of the usual effort and without reinventing the wheel. For this purpose, an IoT platform is the new wheel.

The clayster platform, Interfacing our devices using XMPP,Creating control application.

The Clayster platform

In this chapter, we will redevelop the Controller application that we developed in the previous chapter and call it Controller2. We will, however, develop it as a service to be run on the Clayster Internet of Things platform. In this way, we can compare the work that was required to develop it as a standalone application with the effort that is required to create it as a service running on an IoT platform. We can also compare the results and see what additional benefits we will receive by running our service in an environment where much that is required for a final product already exists.

Downloading the Clayster platform

We will start the download of the Clayster platform by downloading its version meant for private use from http://www.clayster.com/downloads.

Before you are able to download the platform, the page will ask you to fill in a few personal details and a working e-mail address. When the form is filled, an e-mail will be sent to you to confirm the address. Once the e-mail address has been confirmed, a distribution of the platform will be built for you, which will contain your personal information. When this is done, a second e-mail will be sent to you that will contain a link to the distribution along with instructions on how to install it on different operating systems.

STET WOMEN'S COLLEGE

In our examples available for download, we have assumed that you will install Clayster platform on a Windows machine in the C:\Downloads\ClaysterSmall folder. This is not a requirement. If you install it in another folder or on another operating system, you might have to update the references to Clayster Libraries in the source code for the code to compile.

All the information about Clayster, including examples and tutorials, is available in a wiki. You can access this wiki at https://wiki.clayster.com/.

Creating a service project

Creating a service project differs a little from how we created projects in the previous chapters. *Appendix A*, *Console Applications* outlines the process to create a simple console application. When we create a service for a service platform, the executable EXE file already exists. Therefore, we have to create a library project instead and make sure that the target framework corresponds to the version of the Clayster distribution (.NET 3.5 at the time of writing this book). Such a project will generate a dynamic link library (DLL) file. During startup, the Clayster executable file will load any DLL file it finds marked as a *Clayster Module* in its installation folder or any of its subfolders.

Adding references

The Clayster distribution and runtime environment already contains all Clayster libraries. When we add references to these libraries from our project, we must make sure to use the libraries available in the Clayster distribution from the installation folder, instead of using the libraries that we used previously in this book. This will ensure that the project uses the correct libraries. Apart from the libraries used previously, there are a few new libraries available in the Clayster distribution that are new to us, and which we will need:

- Clayster.AppServer.Infrastructure: This library contains the application engine available in the platform. Apart from managing applications, it also provides report tools, cluster support, management support for operators and administrators; it manages backups, imports, exports, localization and various data sources used in IoT, and it also provides rendering support for different types of GUIs, among other things.
- Clayster.Library.Abstract: This library contains a data abstraction layer, and is a crucial tool for the efficient management of objects in the system.
- Clayster.Library.Installation: This library defines the concept of packages.
- Clayster.Library.Meters: This library replaces the Clayster.Library.IoT library used in previous chapters. It contains an abstraction model for things such as sensors, actuators, controllers, meters, and so on.

Apart from the libraries, we will also add two additional references to the project—this time to two service modules available in the distribution, which are as follows:

- Clayster.HomeApp.MomentaryValues: This is a simple service that displays momentary values using gauges. We will use this project to display gauges of our sensor values.
- Clayster.Metering.Xmpp: This module contains an implementation of XMPP on top of the abstraction model defined in the Clayster.Library.Metersz namespace. It does everything we did in the previous chapter and more.

STET WOMEN'S COLLEGE

Making a Clayster module

Not all DLLs will be loaded by Clayster. It will only load the DLLs that are marked as Clayster modules. There are three requirements for a DLL to be considered as a Clayster module:

- The module must be CLS-compliant.
- It must be marked as a Clayster module.
- It must contain a public certificate with information about the developer.

Tip

There are a lot of online services that allow you to create simple self-signed certificates. One such service can be found at www.getacert.com.

All these things can be accomplished through the AssemblyInfo.cs file, available in each .NET project. Enforcing CLS compliance is easy. All you need to do is add the CLSCompliant assembly attribute, defined in the System namespace, as follows:

```
using System;
[assembly: CLSCompliant(true)]
```

This will make sure that the compiler creates warnings every time it finds a construct that is not CLS-compliant.

The two last items can be obtained by adding a public (possibly self-generated) certificate as an embedded resource to the project and referencing it using the Certificate assembly attribute, defined in the Clayster.Library.Installation library, as follows:

```
[assembly: Certificate("LearningIoT.cer")]
```

Note

This certificate is not used for identification or security reasons. Since it is embedded into the code, it is simple to extract. It is only used as a means to mark the module as a Clayster module and provide information about the developer, so that module-specific data is stored appropriately and locally in an intuitive folder structure.

Executing the service

There are different ways in which you can execute a service. In a commercial installation, a service can be hosted by different types of hosts such as a web server host, a Windows service host or a standalone executable host. While the first two types are simpler to monitor and maintain in a production environment, the latter is much easier to work with during development. The service can also be hosted in a cluster of servers.

The small Clayster distribution you've downloaded contains a slightly smaller version of the standalone executable host that can be run on Mono. It is executed in a terminal window and displays logged events. It loads any Clayster modules found in its folder or any of its

STET WOMEN'S COLLEGE

subfolders. If you copy the resulting DLL file to the Clayster installation folder, you can simply execute the service by starting the standalone server from Windows, as follows:

```
Clayster.AppServer.Mono.Standalone
```

If you run the application from Linux, you can execute it as follows:

```
$ sudo mono Clayster.AppServer.Mono.Standalone.exe
```

Using a package manifest

Instead of manually copying the service file and any other associated project files, such as content files, the developer can create a package manifest file describing the files included in the package. This makes the package easier to install and distribute. In our example, we only have one application file, and so our manifest file becomes particularly simple to write. We will create a new file and call it Controller2.packagemanifest. We will write the following into this new file and make sure that it is marked as a content file:

```
<?xml version="1.0" encoding="utf-8" ?>
<ServicePackageManifest
xmlns="http://clayster.com/schema/ServicePackageManifest/v1.xsd">
<ApplicationFiles>
<ApplicationFile file="Controller2.dll"/>
</ApplicationFiles>
</ServicePackageManifest>
```

Tip

You can find more information on how to write package manifest file can be found at https://wiki.clayster.com/mediawiki/index.php?title=Service_Package_Manifest_File.

Now that we have a package manifest file, we can install the package and execute the standalone server from the command line in one go. For a Windows system we can use the following command:

```
Clayster.AppServer.Mono.Standalone –i Controller2.packagemanifest
```

On a Linux system the standalone server can be executed using the following command:

```
$ sudo mono Clayster.AppServer.Mono.Standalone.exe –i Controller2.packagemanifest
```

Before loading the server and executing the particular service, the executable file analyzes the package manifest file and copies all the files to the location where they belong.

Executing from Visual Studio

If you are working with the professional version of Visual Studio, you can execute the service directly from the IDE. This will allow you to debug the code directly from the IDE. To do

STET WOMEN'S COLLEGE

this, you need to open the properties by right-clicking on the project in the Solution Explorer and go to the Debug tab. As a Start Action, choose the Start external program option. There you need to search for the Clayster.AppServer.Mono.Standalone.exe file and enter -i Controller2.packagemanifest on the command line arguments box. Now you can execute and debug the service directly from the IDE.

Configuring the Clayster system

The Clayster system does many things automatically for us that we have manually done earlier. For instance, it maintains a local object database, configures a web server, configures mail settings, connects to an XMPP server, creates an account, registers with a Thing Registry and provisioning server, and so on. However, we need to provide some details manually for this to work. We will do this by editing the Clayster.AppServer.Infrastructure.Settings.xml file.

In Raspberry Pi, we will do this with the following command:

```
$ sudo nano Clayster.AppServer.Infrastructure.Settings.xml
```

The file structure is already defined. All we need to do is provide values for the SmtpSettings element so that the system can send e-mails. We can also take this opportunity to validate our choice of XMPP server in the XmppServerSettings element, which by default is set to thingk.me, and our HTTP server settings, which are stored in the HttpServerSettings and HttpCacheSettings elements.

Tip

You can find more detailed information about how to set up the system at https://wiki.clayster.com/mediawiki/index.php?title=Clayster_Setting_Up_Index.

Using the management tool

Clayster comes with a management tool that helps you to manage the server. This Clayster Management Tool (CMT), can also be downloaded from http://www.clayster.com/downloads.

Apart from the settings file described in the previous section, all other settings are available from the CMT. This includes data sources, objects in the object database, current activities, statistics and reports, and data in readable event logs.

When the CMT starts, it will prompt you for connection details. Enter a name for your connection and provide the IP address of your Raspberry Pi (or localhost if it is running on your local machine). The default user name is Admin, and the default password is the blank password. Default port numbers will also be provided.

Tip

To avoid using blank passwords, the CMT will ask you to change the password after the first login.

STET WOMEN'S COLLEGE

A typical login window that appears when CMT starts will looks like the following screenshot:

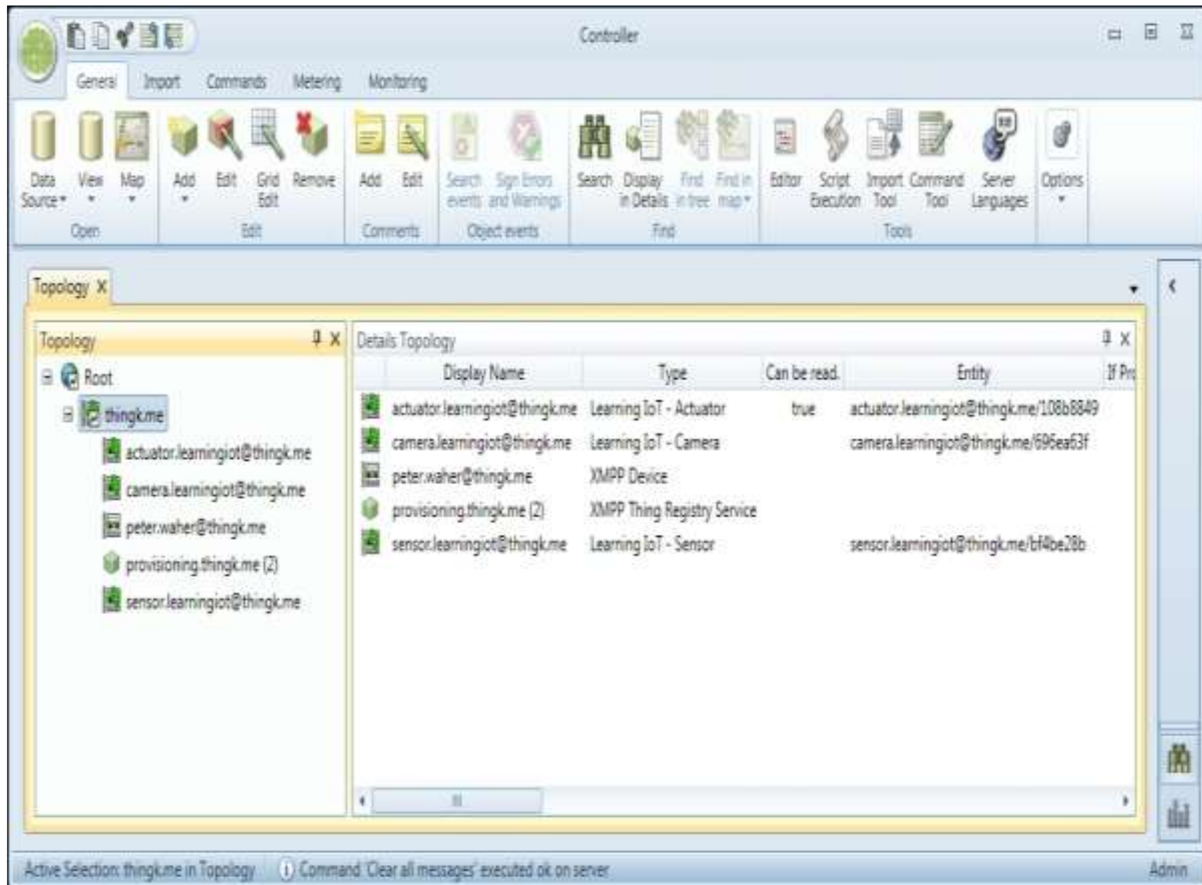
Login window in CMT

Browsing data sources

Most of the configurable data in Clayster is ordered into data sources. These can be either tree-structured, flat or singular data sources. Singular data sources contain only one object. Flat sources contain a list (ordered or unordered) of objects. Tree structured data sources contain a tree structure of objects, where each object in the structure represents a node. The tree-structured data sources are the most common, and they are also often stored as XML files. Objects in such data sources can be edited directly in the corresponding XML file, or indirectly through the CMT, other applications or any of the other available APIs.

When you open the CMT for the first time, make sure that you open the Topology data source. It is a tree-structured data source whose nodes represent IoT devices. The tree structure shows how they are connected to the system. The Root represents the server itself.

In the following example, we can see the system (Root) connected to an XMPP Server (via an account). Through this account, five entities can be accessed (as "friends"). Our sensor, actuator, and camera are available and online (marked in green). Our thing registrar app is a connection, but is not currently online. We are also connected to a Thing Registry and provisioning service. Each node adds its functionality to the system.



Displaying the Topology data source in CMT

In the CMT, you can view, modify, delete, import, and export all objects in these data sources. Apart from the Topology data source, there are a lot of other available data sources. Make sure that you familiarize yourself with them.

Interfacing our devices using XMPP

XMPP is already implemented and supported through the Clayster.Metering.Xmpp module that we mentioned earlier. This module models each entity in XMPP as a separate node in the Topology data source. Connections with provisioning servers and thing registries are handled automatically through separate nodes dedicated to this task. Friendships are handled through simple child creation and removal operations. It can be done automatically through requests made by others or recommendations from the provisioning server, or manually by adding friends in the CMT. All we need to do is provide specialized classes that override base class functionality, and add specific features that are needed.

Creating a class for our sensor

In our project, we will create a class for managing our sensor. We will derive it from the XmppSensor class defined in Clayster.Metering.Xmpp and provide the required default constructor through the following code:

```
public class Sensor : XmppSensor
{
public Sensor()
{
}
```

Each class managed by Clayster.Library.Abstract, such as those used by the Topology data source, must define a TagName and a Namespace property. These are used during import and export to identify the class in question as follows:

```
public override string TagName
{
get { return "IoTSensor"; }
}

public override string Namespace
{
get { return "http://www.clayster.com/learningiot/"; }
}
```

We must also provide a human readable name to the class. Whenever objects of this class are displayed, for instance in the CMT, it is this human readable name that will be displayed, as shown in the following code:

```
public override string GetDisplayableTypeName (Language UserLanguage)
{
return "Learning IoT - Sensor";
}
```

Finding the best class

When the system finds a new device, it needs to know which class best represents that device. This is done by forcing each XMPP device class to implement a Supports method that returns to which degree the class handles said device, based on features and interoperability interfaces reported by the device. The class with the highest support grade is then picked to handle the newly found device.

By using the following code, we will override this method to provide a perfect match when our sensor is found:

```
public override SupportGrade Supports (
XmppDeviceInformation DeviceInformation)
{
if (Array.IndexOf<string> (
DeviceInformation.InteroperabilityInterfaces,
"Clayster.LearningIoT.Sensor.Light") >= 0 &&
```


STET WOMEN'S COLLEGE

```
Array.IndexOf<string> (
DeviceInformation.InteroperabilityInterfaces,
"Clayster.LearningIoT.Sensor.Motion") >= 0)
{
return SupportGrade.Perfect;
}
else
return SupportGrade.NotAtAll;
}
```

Subscribing to sensor data

Manual readout of the sensor is already supported by the XmppSensor class. This means you can already read data from the sensor from the CMT, for instance, as it is. However, this is not sufficient for our application. We want to subscribe to the data from the sensor. This subscription is application-specific, and therefore must be done by us in our application. We will send a new subscription every time the sensor reports an online or chat presence. The XmppSensor class will then make sure that the subscription is sent again if the data is not received accordingly. The subscription call is similar to the one we did in the previous chapter. The subscription call is sent using the following code:

```
protected override void OnPresenceChanged (XmppPresence Presence)
{
if (Presence.Status == PresenceStatus.Online ||
Presence.Status == PresenceStatus.Chat)
{
this.SubscribeData (-1, ReadoutType.MomentaryValues,
new FieldCondition[] {
FieldCondition.IfChanged ("Temperature", 0.5),
FieldCondition.IfChanged ("Light", 1),
FieldCondition.IfChanged ("Motion", 1)
}, null, null, new Duration (0, 0, 0, 0, 1, 0), true,
this.NewSensorData, null);
}
}
```

Interpreting incoming sensor data

Interpreting incoming sensor data is done using the Clayster platform in a way that is similar to what we did using the Clayster.Library.IoT library in the previous chapters. We will start by looping through incoming fields:

```
private void NewSensorData (object Sender, SensorDataEventArgs e)
{
FieldNumeric Num;
FieldBoolean Bool;
double? LightPercent = null;
bool? Motion = null;
```



```
if(e.HasRecentFields)
```

STET WOMEN'S COLLEGE

```
{
foreach(Field Field in e.RecentFields)
{
switch(Field.FieldName)
{
```

There is one added advantage of handling field values when we run them on the Clayster platform: we can do unit conversions very easily. We will illustrate this with the help of an example, where we handle the incoming field value - temperature. First, we will try to convert it to Celsius. If successful, we will report it to our controller application (that will soon be created):

```
case "Temperature":
if ((Num = Field as FieldNumeric) != null)
{
Num = Num.Convert ("C");
if (Num.Unit == "C")
Controller.SetTemperature (Num.Value);
}
break;
```

Tip

There is a data source dedicated to unit conversion. You can create your own unit categories and units and determine how these relate to a reference unit plane, which is defined for each unit category. Unit conversions must be linear transformations from this reference unit plane.

We will handle the Light and Motion values in a similar way. Finally, after all the fields have been processed, we will call the Controller application and ask it to check its control rules:

```
if (LightPercent.HasValue && Motion.HasValue)
Controller.CheckControlRules (
LightPercent.Value, Motion.Value);
}
}
```

Our Sensor class will then be complete.

Creating a class for our actuator

If implementing support for our Sensor class was simple, implementing a class for our actuator is even simpler. Most of the actuator is already configured by the XmppActuator class. So, we will first create an Actuator class that is derived from this XmppActuator class. We will provide it with a TagName that will return "IoTActuator" and the same namespace that the Sensor class returns. We will use Learning IoT – Actuator as a displayable type name. We will also override the Supports method to return a perfect response when the corresponding interoperability interfaces are found.

STET WOMEN'S COLLEGE

Customizing control operations

Our Actuator class is basically complete. The XmppActuator class already has support for reading out the control form and publishing the available control parameters. This can be tested in the CMT, for instance, where the administrator configures control parameters accordingly.

To make control of the actuator a bit simpler, we will add customized control methods to our class. We already know that the parameters exist (or should exist) since the corresponding interoperability interfaces (contracts) are supported.

We will begin by adding a method to update the LEDs on the actuator:

```
public void UpdateLeds(int LedMask)
{
this.RequestConfiguration ((NodeConfigurationMethod)null, "R_Digital Outputs", LedMask,
this.Id);
}
```

The RequestConfiguration method is called to perform a configuration. This method is defined by Clayster.Library.Meters namespace, and can be called for all configurable nodes in the system. Configuration is then performed from a context that is defined by the node. The XmppActuatorclass translates this configuration into the corresponding set operation, based on the data type of the parameter value.

The first parameter contains an optional callback method that is called after the parameter has been successfully (or unsuccessfully) configured. We don't require a callback, so we will only send a null parameter value. The second parameter contains the name of the parameter that needs to be configured. Local configurable parameters of the XmppActuator class differ from its remote configurable parameters, which are prefixed by R_. The third parameter value is the value that needs to be configured. The type of value to send here depends on the parameter used. The fourth and last parameter is a subject string that will be used when the corresponding configuration event is logged in the event log.

Tip

You can find out which configurable parameters are available on a node by using the CMT.

In a similar fashion, we will add a method for controlling the alarm state of the actuator and then our Actuator class will be complete.

Creating a class for our camera

In essence, our Camera class does not differ much from our Sensor class. It will only have different property values as well as a somewhat different sensor data subscription and field-parsing method. Interested readers can refer to the source code for this chapter.

STET WOMEN'S COLLEGE

Creating our control application

We are now ready to build our control application. You can build various different kinds of applications on Clayster. Some of these have been listed as follows:

- 10-foot interface applications: These applications are suitable for TVs, smart phones and tablets. They are created by deriving from the Clayster.AppServer.Infrastructure.Application class. The name emerged from the requirement that the application should be usable from a distance of 10 feet (about 3 meters), like a television set. This, for instance, requires large fonts and buttons, and no windows. The same interface design is suitable for all kinds of touch displays and smart phones.
- Web applications: These applications are suitable for display in a browser. These are created by deriving from the Clayster.AppServer.Infrastructure.WebApplication class. The thingk.me service is a web application running on the Clayster platform.
- Non-visible services: These services can be implemented, by the Clayster.Library.Installation.Interfaces.IPluggableModule interface.
- Custom views: These views for integration with the CMT can be implemented by deriving from Clayster.Library.Layout.CustomView.

The first two kinds of applications differ in one important regard: Web applications are assumed to be scrollable from the start, while 10-foot interface applications have to adhere to a fixed-size screen.

Understanding rendering

When creating user interfaces in Clayster, the platform helps the developer by providing them with a powerful rendering engine. Instead of you having to provide a complete end user GUI with client-side code, the rendering engine creates one for you dynamically. Furthermore, the generated GUI will be created for the client currently being used by the user. The rendering engine only takes metadata about the GUI from the application and generates the GUI for the client. In this way, it provides a protective, generative layer between application logic and the end user client in much the same way as the object database handles database communication for the application, by using metadata available in the class definitions of objects that are being persisted.

The rendering pipeline can be simplistically described as follows:

1. The client connects to the server.
2. An appropriate renderer is selected for the client, based on protocol used to connect to the server and the capabilities of the client. The renderer is selected from a list of available renderers, which themselves are, to a large extent, also pluggable modules.
3. The system provides a Macro-layout for the client. This Macro-layout is devoid of client-specific details and resolutions. Instead, it consists of a basic subdivision of available space. Macro-layouts can also be provided as pluggable modules. In the leaf nodes of this Macro-layout, references are made either explicitly or implicitly to services in the system. These services then provide a Micro-layout that is used to further subdivide the available space. Micro-layout also provides content for the corresponding area.

STET WOMEN'S COLLEGE

Tip

More information about Macro-layout and Micro-layout can be found here https://wiki.clayster.com/mediawiki/index.php?title=Macro_Layout_and_Micro_Layout.

1. The system then provides a theme, which contains details of how the layout should be rendered. Themes can also be provided as pluggable modules.
2. The final interactive GUI is generated and sent to the client. This includes interaction logic and support for push notification.

Defining the application class

Since we haven't created a 10-foot interface application in the previous chapters, we will create one in this chapter to illustrate how they work. We will start by defining the class:

```
public class Controller : Application
{
public Controller ()
{
}
```

Initializing the controller

Much of the application initialization that we did in the previous chapters has already been taken care of by the system for us. However, we will still need a reference to the object database and a reduced mail settings class. Initialization is best done by overriding the OnLoaded method:

```
internal static ObjectDatabase db;
internal static MailSettings mailSettings;

public override void OnLoaded ()
{
db = DB.GetDatabaseProxy ("TheController");
mailSettings = MailSettings.LoadSettings ();

if (mailSettings == null)
{
mailSettings = new MailSettings ();
mailSettings.From = "Enter address of sender here.";
mailSettings.Recipient = "Enter recipient of alarm mails here.";
mailSettings.SaveNew ();
}
}
```

Adding control rules

The control rules we define for the application will be the same as those used in previous chapters. The only difference here is that we don't need to keep track of the type or number of

STET WOMEN'S COLLEGE

devices that are currently connected to the controller. We can simply ask the Topology data source to return all the items of a given type, as follows:

```
if (!lastAlarm.HasValue || lastAlarm.Value != Alarm)
{
lastAlarm = Alarm;
UpdateClients ();

foreach (Actuator Actuator in Topology.Source.GetObjects(
typeof(Actuator), User.AllPrivileges))
Actuator.UpdateAlarm (Alarm);

if (Alarm)
{
Thread T = new Thread (SendAlarmMail);
T.Priority = ThreadPriority.BelowNormal;
T.Name = "SendAlarmMail";
T.Start ();
}
```

The second parameter in the GetObjects call is a user object. It is possible to limit access to objects in a data source based on access privileges held by the role of the user. A predefined user having all access rights (User.AllPrivileges) assures us that we will get all the objects of the corresponding type. Also, note that we made a call to an UpdateClients method. We will define this method later. It will ensure that anything that causes changes in the GUI is pushed up to the connected end users.

Tip

Users, roles, and privileges are three separate data sources that are available in Clayster. You can manage these in the CMT if you have sufficient privileges. Nodes in the Topology data source can require visible custom privileges. Edit the corresponding nodes to set such custom privileges. This might allow you to create an environment with compartmentalized access to Topology data source and other data sources.

Understanding application references

Macro-layouts provided by the system can reference applications in the system in different ways:

- Menu reference: A menu reference consists of a reference to the application together with an instance name string. Micro-layout for a menu reference is fetched by calling the OnShowMenu method on the corresponding application. There are three types of menu references:
  -
    1. Standard menu reference: This appears in normal menus.
    2. Custom menu reference: This is a custom area of custom size. It can be considered a widget. In Clayster, such a widget is called a brieflet.

STET WOMEN'S COLLEGE

3. Dynamic selection reference: This is a selection area that can display detailed information about a selected item from a selected application on the screen.

- Dialog reference: A dialog reference consists of a reference to an application, together with an instance name string and a dialog name string. Micro-layout for a dialog reference will be fetched by calling the OnShowDialog method on the corresponding application.

Defining brieflets

In our example, we will only use custom menu references, or the so-called brieflets. We don't need to create menus for navigation or dialogs containing user interaction. Everything that we need to display will fit into one simple screen. First, we will tell the system that the application will not be visible in regular menus:

```
public override bool IsShownInMenu(IsShownInMenuEventArgs e)
{
return false;
}
```

This is the method in which the application can publish standard menu references. We will then define the brieflets that we want to publish. This will be done by overriding the GetBrieflets method, as follows:

```
public override ApplicationBrieflet[] GetBrieflets (
GetBriefletEventArgs e)
{
return new ApplicationBrieflet[] {
new ApplicationBrieflet ("Temperature",
"Learning IoT - Temperature", 2, 2),
new ApplicationBrieflet ("Light",
"Learning IoT - Light", 2, 2),
new ApplicationBrieflet ("Motion",
"Learning IoT - Motion", 1, 1),
new ApplicationBrieflet ("Alarm",
"Learning IoT - Alarm", 1, 1)
};
}
```

The first parameter in each brieflet definition is the instance name identifying the brieflet. The second parameter is a human readable string that is used when a list of available brieflets is presented to a human user. The last two parameters correspond to the size of the brieflet. The unit that is used is the number of "squares" in an imaginary grid. A menu item in a touch menu can be seen as a 1 x 1 square.

Displaying a gauge

All our brieflets are customized menu items. So, to display something in one of our brieflets, we just need to return the corresponding Micro-layout by overriding

STET WOMEN'S COLLEGE

the OnShowMenu method. In this example, we want to start by returning Micro-layout for the temperature brieflet:

```
public override MicroLayout OnShowMenu (ShowEventArgs e)
{
switch (e.InstanceName)
{
case "Temperature":
```

Micro-layout can be defined either by using XML or dynamically through code, where each XML element corresponds to a class with the same name. We will use the second approach since it is easier to create a dynamic Micro-layout this way. We will use the applicationClayster.HomeApp.MomentaryValues, available in the distribution, to quickly draw a bitmap image containing a gauge displaying our sensor value. This is shown in the following code snippet:

```
MicroLayoutElement Value;
System.Drawing.Bitmap Bmp;

if (temperatureC.HasValue)
{
Bmp = Clayster.HomeApp.MomentaryValues.Graphics.GetGauge (15, 25,
temperatureC.Value, "°C", GaugeType.GreenToRed);
Value = new ImageVariable (Bmp);
}
else
Value = new Label ("N/A");
```

Tip

Bitmap content can be displayed using either ImageVariable or ImageConstant (or any of its descendants). We have used ImageVariable in this example and we will use ImageConstant to display camera images.

Constant images also provide a string ID, which identifies the image. Using this ID, the image can be cached on the client, and it will be fetched from the server only if the client does not already have the image in its cache. This requires less communication resources, but may induce flicker when the image changes and the new image is not available in the cache and while it is being loaded. ImageVariable supposes the image to be new for every update. It requires more communication resources, but provides updates without flicker since the image data is embedded into the frame directly. You can try the two different methods separately to get a feel for how they work.

When we get the gauge—or the label if no value is available—we will return the Micro-layout. Remember that Macro-layout and Micro-layout work by subdividing the available space, rather than placing controls on a form. In our case, we will divide the available space into two rows of relative heights 1:3, the top one containing a header and the lower one the gauge or label.

STET WOMEN'S COLLEGE

```
return new MicroLayout (new Rows (
new Row (1, HorizontalAlignment.Center,
VerticalAlignment.Center,
Paragraph.Header1 ("Temperature")),
new Row (3, HorizontalAlignment.Center,
VerticalAlignment.Center, Value)));
```

The brieflet showing the light gauge is handled in exactly the same way.

Displaying a binary signal

The layouts for the binary motion and alarm signals are laid out in a manner similar to what we just saw, except the size of the brieflet is only 1 x 1:

```
case "Motion":
Value = this.GetAlarmSymbol (motion);

return new MicroLayout (new Rows (
new Row (1, HorizontalAlignment.Center,
VerticalAlignment.Center,
Paragraph.Header1 ("Motion")),
new Row (2, HorizontalAlignment.Center,
VerticalAlignment.Center, Value)));
```

The same code is required for the alarm signal as well. A binary signal can be displayed by using two constant images. One represents 0 or the "off" state, and the other represents 1 or the "on" state. We will use this method in binary brieflets by utilizing two images that are available as embedded resources in the application Clayster.HomeApp.MomentaryValues, to illustrate the point:

```
private MicroLayoutElement GetAlarmSymbol(bool? Value)
{
if (Value.HasValue)
{
if (Value.Value)
{
return new MicroLayout (new ImageMultiResolution (
new ImageConstantResource (
"Clayster.HomeApp.MomentaryValues." +
"Graphics._60x60.Enabled." +
"blaljus.png", 60, 60),
new ImageConstantResource (
"Clayster.HomeApp.MomentaryValues." +
"Graphics._45x45.Enabled." +
"blaljus.png", 45, 45)));
}
else
return new MicroLayout (new ImageMultiResolution (
new ImageConstantResource (
"Clayster.HomeApp.MomentaryValues." +
```


STELLA WOMEN'S COLLEGE

```
"Graphics._60x60.Disabled." +
"blaljus.png", 60, 60),
new ImageConstantResource (
"Clayster.HomeApp.MomentaryValues." +
"Graphics._45x45.Disabled." +
"blaljus.png", 45, 45)));
}
}
else
return new Label ("N/A");
}
```

Note

Micro-layout supports the concept of multiresolution images. By providing various options, the renderer can choose the image that best suits the client, given the available space at the time of rendering.

Pushing updates to the client

It's easy to push updates to a client. First, you need to enable such push notifications. This can be done by enabling events in the application as follows. By default, such events are disabled:

```
public override bool SendsEvents
get { return true; }
```

Each client is assigned a location. For web applications, this location is temporary. For 10-foot interfaces, it corresponds to a Location object in the geo-localized Groups data source. In both cases, each location has an object ID or OID. To forward changes to a client, an application will raise an event providing the OID corresponding to the location where the change should be executed. The system will handle the rest. It will calculate what areas of the display are affected, render a new layout and send it to the client.

Tip

All areas of the screen corresponding to the application will be updated on the corresponding client. If you have multiple brieflets being updated asynchronously from each other, it is better to host these brieflets using different application classes in the same project. This avoids unnecessary client updates. In our code, we will divide our brieflets between three different applications, one for sensor values, one for camera images and one for test command buttons.

Push notifications in our application are simple. We want to update any client who views the application after a sensor value is updated, regardless of the location from which the client views the application. To do this, we first need to keep track of which clients are currently viewing our application. We define a Dictionary class as follows:


```
private static Dictionary<string,bool> activeLocations = new Dictionary<string, bool> ();
```

STET WOMEN'S COLLEGE

We will populate this Dictionary class with the object IDs OID of the location of the clients as they view the application:

```
public override void OnEventNotificationRequest (Location Location)
{
lock(activeLocations)
{
activeLocations [Location.OID] = true;
}
}
```

And depopulate it as soon as a client stops viewing the application:

```
public override void OnEventNotificationNoLongerRequested (Location Location)
{
lock (activeLocations)
{
activeLocations.Remove (Location.OID);
}
}
```

To get an array of locations that are currently viewing the application, we will simply copy the keys of this dictionary into an array that can be safely browsed:

```
public static string[] GetActiveLocations ()
{
string[] Result;
lock (activeLocations)
{
Result = new string[activeLocations.Count];
activeLocations.Keys.CopyTo (Result, 0);
}
return Result;
}
```

Updating clients is now easy. Whenever a new sensor value is received, we will call the UpdateClients method, which in turn will register an event on the application for all clients currently viewing it. The platform will take care of the rest:

```
private static string appName = typeof(Controller).FullName;

private static void UpdateClients ()
{
foreach (string OID in GetActiveLocations()) EventManager.RegisterEvent (appName, OID);
}
```

Completing the application

The source code for our project contains more brieflets that are defined in two more application classes. The CamStorage class contains three brieflets that show the last three

STET WOMEN'S COLLEGE

camera images that were taken. They use ImageConstant to display the image to the client. The application also pushes updates to clients in the same way in which the Controller class does. However, by putting the brieflets in a separate application, we can avoid updating the entire screen when a new camera image is taken or when sensor values change.

A third application class named TestApp publishes two small brieflets, each containing a Test button that can be used to test the application. It becomes quickly apparent if the sensor is connected and works, since changes to sensor values are followed by changes in the corresponding gauges. To test the actuator, one brieflet publishes a Test button. By clicking on it, you can test the LED and alarm outputs. A second brieflet publishes a Snapshot button. By clicking this button you can take a photo, if a camera is connected, and update any visible camera brieflets.

Configuring the application

We can now try the application. We will execute the application as described earlier. The first step is to configure the application so that the devices become friends and can interchange information with each other. This step is similar to what we did in the previous chapter. You can either configure friendships manually or use thingk.me to control access permissions between the different projects and the new service.

Note that the application will create a new *JID* for itself and register it with the provisioning server. It will also log a *QR* code to the event log, which will be displayed in the terminal window. This QR code can be used to claim ownership of the controller.

Tip

Remember to use the CMT application to monitor the internal state of your application when creating friendships and trying readouts and control operations. From the CMT, you can open line listeners to monitor actual communication. This can be done by right-clicking the node in the Topology data source that represents the XMPP server.

Viewing the 10-foot interface application

After starting the Clayster platform with our service, we can choose various ways to view the application. We can either use a web browser or a special Clayster View application. For simplicity's sake, we'll use a web browser. If the IP address of our controller is 192.168.0.12, we can view the 10-foot interface at http://192.168.0.12/Default.ext?ResX=800&ResY=600&HTML5=1&MAC=000000000001&SimDisplay=0&SkipDelay=1.

Tip

For a detailed description on how to form URLs for 10-foot interface clients, see https://wiki.clayster.com/mediawiki/index.php?title=Startup_URLs.

There are various ways to identify the location object to which the client corresponds. This identification can be done by using the client's IP address, MAC address, login user name, certificate thumbprint, XMPP address, or a combination of these.

STET WOMEN'S COLLEGE

Note

The ClaysterSmall distribution comes with a Groups data source containing one location object identified by MAC address 000000000001. If you are using other identification schemes, or a client that reports a true MAC address, then the corresponding location object must be updated. For more information on this go to https://wiki.clayster.com/mediawiki/index.php?title=Groups_-_Location.

You can also configure the system to automatically add Location objects to your Groups data source. This would allow automatic installation of new client devices.

To be able to configure the screen the way we want, we will enter the Settings menu, click on the Layout menu item and select the No Menu 5x4 option. This will clear the display and allow you to experiment with placing brieflets over the entire screen using a 5x4 grid of squares. Simply click on the user-defined layout on an area that does not have a brieflet, and you can select which brieflet you want to display there.

After arranging the brieflets the way we want, the screen might look something like the following screenshot. Gauges, binary signals, and camera images will be automatically pushed to the client, and from this interface we can see both the current state of the controller as well as test all the parts of the system.

STET WOMEN'S COLLEGE