Авторы: Горбунцов М.п. и Хатнянский м.е.



Интернет Вещей

В этом пособии вы поймете, что такое IoT (Интернет вещей) в обширном понимании этой компетенции. Для чего нужна данная компетенция, где применяется и ее перспективы.

Интернет Вещей

Авторы: Горбунцов М.п. и Хатнянский м.е.

авторы

Мы студенты, учащиеся в Кузнецком индустриальном техникуме по специальности «Информационные системы и программирования», участники всероссийского чемпионата «World Skills Russia 2020» по компетенции «IoT – Интернет вещей».

## Содержание

При составлении данного пособия были приложены все силы для передачи нашего опыта, чтобы при изучении пособия не было непонятных сложных терминов и скучной, заядлой документации, но в небольшом объеме она все равно будет. Только по факту нужный для вас материал для быстрого понимания и вливание вас в данную компетенцию. Большую часть терминов и значений мы будем описывать простым языком, для того чтобы человек даже не знающий программирования смог разобраться и применить свои знания на практике.

## Предисловие

Наша подготовка проходила целый год, но это не означает, что вам придется столько же изучать данное пособие чтобы понять, как работает «Интернет вещей» и платформа [ThingWorx](https://www.ptc.com/ru/products/iiot/thingworx-platform). В краткие сроки вы сможете понять, как устроена данная компетенция и что нужно изучить дополнительно чтобы быть «ассом» в данном направлении. Если опираться на опыт авторов, то хватит чуть больше месяца для полного понимания компетенции и платформы на которой построено данное пособие. Если вы полный чайник в программирование, то это не препятствует вам изучать данный материал.

# Часть 1. IoT – Интернет вещей.

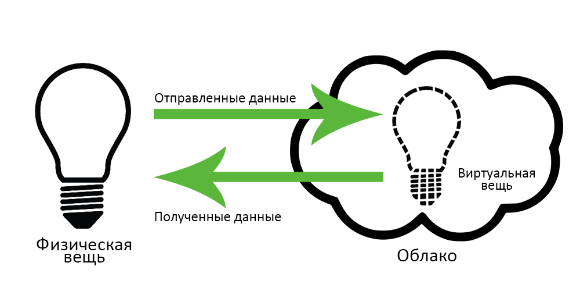
### Глава 1. Общее понимание IoT

Определений Интернета вещей очень много. Мы под Интернетом вещей будем понимать единую сеть, соединяющую окружающие нас объекты реального мира и виртуальные объекты.

Что же такое Интернет вещей?

По одному из определений, с точки зрения IoT, *«вещь» – любой реальный или виртуальный объект, который существует и перемещается в пространстве и времени, может быть однозначно определен.*

Не совсем понятно, что такое IoT, да? Давайте разберем данное определение на более простой «язык»? Для понимания возьмем самую простую вещь, которая имеет доступ к интернету. У нас это будет WiFi – лампочка, вы же можете придумать любую другую вещь, сильного различия не будет. Лампочка имеет доступ к интернету и имеет ряд датчиков, которые отправляют данные о своем состоянии (подключена ли она к интернету, вкл \ выкл, степень освещенности).

Чтобы разобраться дальше, нужно понять, что такое облако в сфере IoT. В общем понимании облако – это онлайн – хранилище, в котором данные хранятся на многочисленных распределённых в сети серверах, но облако уже давно не только хранит данные, но умеет обрабатывать и отправлять результат вычислений пользователю. В нашем случае в облаке создается виртуальная копия физической модели устройства, которая принимает все поступающие данные от реального объекта. Пользователь может изменять значение в облаке и возвращать их на физическую модель устройства, тем самым непосредственно управлять объектом через интернет с любого устройства.

Теперь вернемся к нашей лампочке. Подключая ее к сети интернет через Wi-Fi, ваш роутер автоматически становится сервером-облаком, с помощью которого вы можете управлять лампочкой, физическая модель автоматически создает свою копию в облаке и позволяет к ней обращаться через дополнительное ПО. Все программы управление построены на одной и тоже логике, получи – обработай – отправь.

Это основа понимания IoT, как взаимодействуют объекты между собой. Полная реализация и управление объектами заключается в работе с готовыми облачными платформами или специализированными на IoT. Платформы могут быть как уже готовыми к использованию, так и написанные вручную с помощью [API](https://habr.com/ru/post/464261/) (набор классов, процедур, функций, структур или констант) физических вещей.

IoT – *«концепция пространства, в котором все из аналогового и цифрового миров может быть совмещено – это переопределит наши отношения с объектами, а также свойства и суть самих объектов. »*

Роб Ван Краненбург.

### Глава 2. Зачем нужен интернет вещей

Например, компания Samsung выпустила холодильник Family Hub, помогающий заказывать продукты с доставкой, не выходя из кухни: с помощью сенсорного экрана на дверце холодильника. На него можно вывести каталог магазина и выбрать товары. Холодильник может отслеживать сроки годности каждого продукта и напоминать о них хозяевам. И при каждом закрытии дверцы Family Hub делает фотографию своего содержимого и отправляет ее хозяину на смартфон. Так что не приходится вспоминать, есть ли дома молоко, йогурт или помидоры — достаточно заглянуть в телефон!

Китайская компания Xiaomi выпускает ряд устройств для умных домов. Это и смарт-электрика (розетки, выключатели, удлинители), и беспроводные контроллеры для управления другими гаджетами, и всевозможные датчики — движения, протечки воды, открывания дверей и окон, температуры, влажности и давления. Для большинства девайсов можно настроить сценарии поведения. К примеру, кнопка Xiaomi Smart Wireless Switch предлагает задать разные действия при одиночном щелчке, двойном или долгом нажатии, а Xiaomi Mi Magic Cube Controller, выполненный в виде кубика, можно повернуть, встряхнуть, постучать или сдвинуть — в зависимости от действия он умеет выполнять шесть команд, которые можно запрограммировать самостоятельно. Кнопка дверного звонка пришлет вам на смартфон сообщение, если кто-то придет к вам в ваше отсутствие, а при наличии камеры — еще и покажет фотографию гостя.

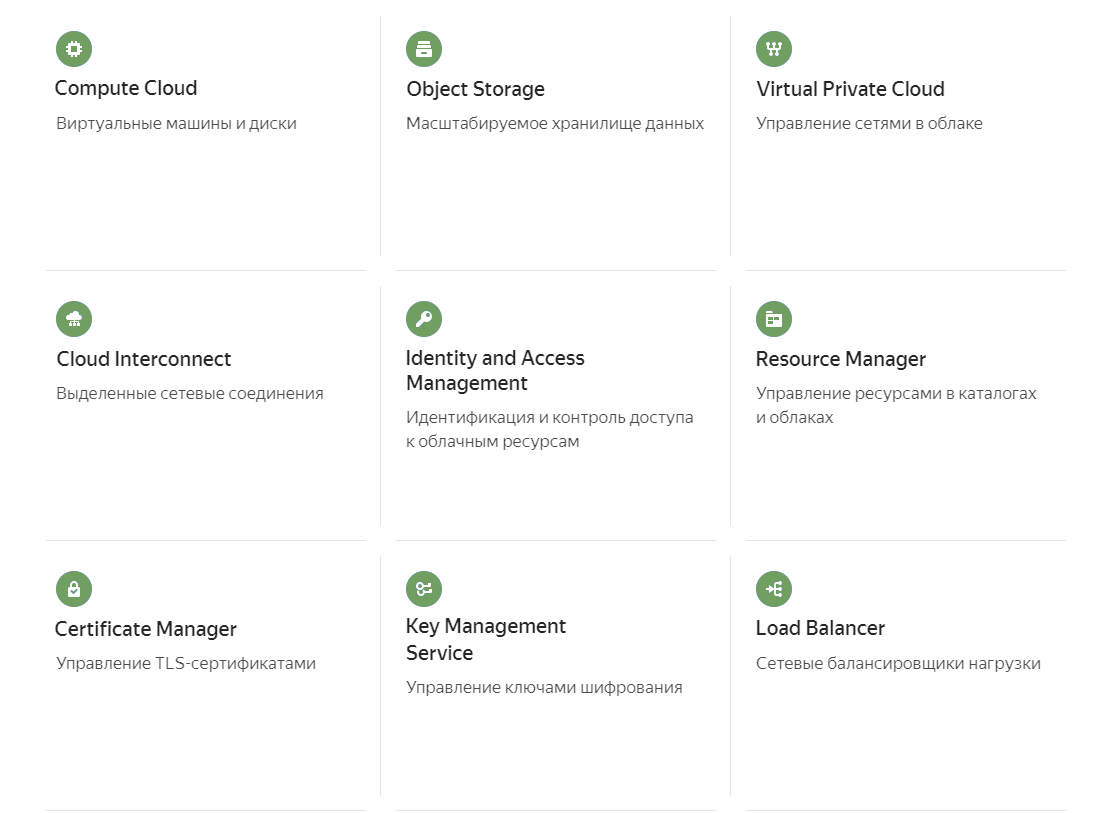
# Часть 2. Облачные платформы

### Глава 1. Общее понимание

На сегодняшний день, облачные технологии используют инструменты и технологии, обеспечивающие создание собственных облачных приложений и управления жизненным циклом разработки программного обеспечения. Различные среды разработки позволяют строить отказоустойчивые и безопасные системы, а также, выполнять многочисленные бессерверные вычисления, учитывая интернет вещей.

Современные облачные сервисы содержат в себе виртуальные машины и диски, масштабируемые хранилища данных, управление сетями в облаке, различные выделенные сетевые соединения, управление ресурсами в каталогах и облаках, идентификацию и контроль доступа к ресурсам, сбор и визуализация метрик, разнообразную защиту от DDoS-атак, управление базой данных SQL, обмен сообщениями между приложениями, запуск вашего кода в виде функции, интеграции с различными сервисами при помощи API и многие другие решения для интернета вещей.

Самыми популярными облачными платформами являются: Yandex.Cloud, Microsoft Azure, Mail.RU CLOUD, PTC(ThingWorx).

Простыми словами, облачные технологии должны обеспечивать пользователю удобную и безопасную среду разработки, для создания взаимодействия с данными и развёртывание своих сервисов.

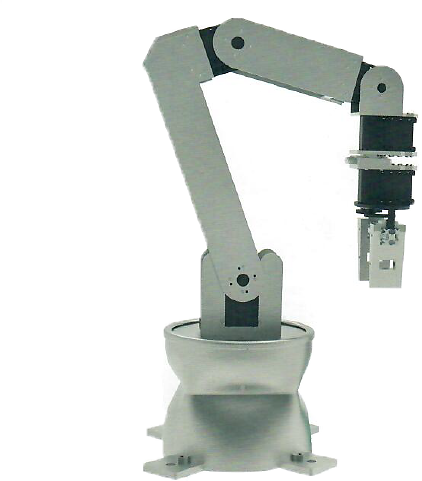
### Глава 2. PTC (ThingWorx)

ThingWorx это облачная платформа компании PTC предназначенная для решение задач которые требуются решения IoT. Данная платформа позволяет решить задачи по типу: удаленного мониторинга, обслуживания и оптимизации. Создание решений для IoT одна из проблемных задач, поэтому была разработана платформа ThingWorx для сокращения количества препятствий. Вы сможете создать пилотный проект для решений кооперативного масштаба с помощью готовых приложений и средств разработки.

Все это скучная нужная документация необходима для общего понимания. Если объяснять кратко, то ThingWorx – это платформа на которой вы сможете создавать свое решение поставленной задачи в плане IoT. Данная платформа содержит в себе решения всех поставленных вами задач. Мы будем работать именно с ThingWorx, так как данная платформа очень легка в освоении и не требует особенных знаний для создания собственных решений IoT.

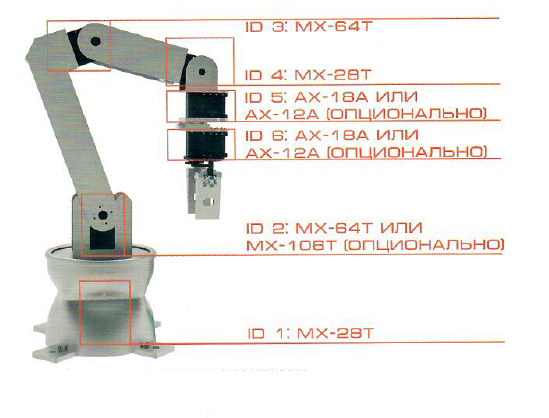
# Часть 3. Знакомство с оборудованием

### Глава 1. Образовательные манипуляторы РТК

 Учебно-лабораторный РТК с угловой кинематикой представляет собой один из самых популярных манипуляторов с точки зрений кинематической схемы. Такую популярность обеспечивает расположение сервоприводов в центре каждого сочленения (соединение) манипулятора, что позволяет рассчитывать математическую модель используя геометрическую схему – треугольника. В зависимости от требования, данный манипулятор может быть дооснащён системами тех. зрения, как моно, так и стерео, а также дополнительными системами захвата, программным пакетом для управления из-под ОС ROS и т.п.



Манипуляционные РТК с плоско-параллельной кинематикой зачастую применяются для перемещения рабочего инструмента параллельно горизонтальной плоскости в цилиндрической системе координат относительно основания. Важным преимуществом данной кинематики является повышенная грузоподъемность на схвате, приобретенной за счет облегчения конструкции манипулятора ближе к схвату, благодоря расположению приводов в основании. В большенстве случаев подобные манипуляторы применяются для выполнения задач по перемещению грузов большой массы в машиностроении или логистики.

Все выше описанное – это техническая документация, обясняя проще у данных роботов есть сервоприводы (моторы), которые позволяют роботу перемещаться по оси X/Y. Так же они могут схватывать предметы при помощи специальной насадки разных видов (ваккум или с плоским схватом). Роботы могут оснощаться дополнительным оборудованием (камерой, доп. ПО).

Все это основное описание роботов.

Помимо этого, есть значение параметров для мониторинга и управление робота:

|  |  |  |
| --- | --- | --- |
| Параметр | Свойство | Ключ |
| Status (статус робота) (int – целочисленный тип)  *Этот параметр не задается, а только получает данные с робота* | 0 – не выполняет действие  1 – выполняет действие | s |
| CurrentCommandNumber (текущий номер команды) (int) | Начинается отсчет команд с 0 до неопределённого числа команд. Прибавляется на 1 от каждого выполненного действия | N |
| LastCommandNumber (int)  *Этот параметр не задается, а только получает данные с робота* | Последняя выполненная команда | n |
| PositionX (int) – параметр перемещение робота по оси Х. | 100 – 300, данный параметр имеет ограничения, заданные в прошивке робота. (перед тем как задавать значение ознакомьтесь с ограничением вашего манипулятора) | X |
| PositionY (int) – параметр перемещение робота по оси Y. | 120 – 280, данный параметр имеет ограничения, заданные в прошивке робота. (перед тем как задавать значение ознакомьтесь с ограничением вашего манипулятора) | Y |
| RotationT (int) – параметр перемещение робота по оси T. | 10 – 300, данный параметр имеет ограничения, заданные в прошивке робота. (перед тем как задавать значение ознакомьтесь с ограничением вашего манипулятора) | T |
| Gripper (int) – параметр отвечающий за отпускание руки захвата | 0 – поднять гриппер  1 – опустить гриппер | G |
| Vaccum (int) – параметр отвечающий за захват детали | 0 – отпустить деталь  1 – схватить деталь | V |

### Глава 2. Удаленный терминал

Удаленный терминал — это специальное оборудование, предназначенное для удаленного доступа к производственной линии. На нем имеется 4 кнопки и 4 лампы индикации работы производства. Кнопки обозначены цветом соответствуя нужному действию, которое задается пользователем. Большая красная кнопка (DeadManSwitch) всегда отвечает за аварийную остановку работы производства. Обозначение кнопок и индикации:

1. Зеленая кнопка – начало автоматической сборки;
2. Красная кнопка – пауза работы производства;
3. Желтая кнопка – стоп работы производства и ожидания новых команд.
4. Красная лампочка – аварийная остановка;
5. Синяя лампочка – производство активно;
6. Зеленая лампочка – ожидание команды;
7. Желтая лампочка – парковка.

Параметры и свойства удаленного терминала (RemoteTerminal):

|  |  |  |
| --- | --- | --- |
| Параметр | Свойство | Ключ |
| DeadManSwitch (int) – аварийная остановка производства. | 0 – кнопка не активна  1 – кнопка активна | d |
| Button1 – первая кнопка | 0 – кнопка не активна  1 – кнопка активна | b1 |
| Button2 – первая кнопка | 0 – кнопка не активна  1 – кнопка активна | b2 |
| Button3 – первая кнопка | 0 – кнопка не активна  1 – кнопка активна | b3 |
| Light1– синяя лампочка | 0 – лампа не активна  1 – лампа активна | L1 |
| Light2– красная лампочка | 0 – лампа не активна  1 – лампа активна | L2 |
| Light3– оранжевая лампочка | 0 – лампа не активна  1 – лампа активна | L3 |
| Light4– зеленая лампочка | 0 – лампа не активна  1 – лампа активна | L4 |

### Глава 3. Светосигнальная лампа

Светосигнальная лампа (TrafficLight) – это специальное оборудование, предназначенное для индикации хода работы производственной ячейки.  
Сигнальная лампа имеет 4 цвета соответственно кодировки:

* Красный – аварийная ситуация;
* Синий – выполнение команд;
* Зеленый - ожидание команды;
* Желтый – парковка, безопасное место для обслуживания.

|  |  |  |
| --- | --- | --- |
| Параметр | Свойство | Ключ |
| Light1– синяя лампочка | 0 – лампа не активна  1 – лампа активна | L1 |
| Light2– красная лампочка | 0 – лампа не активна  1 – лампа активна | L2 |
| Light3– оранжевая лампочка | 0 – лампа не активна  1 – лампа активна | L3 |
| Light4– зеленая лампочка | 0 – лампа не активна  1 – лампа активна | L4 |

Программное расположение лампы таково: Голубой (Light1), Красный (Light2), Оранжевый (Light3), Зеленый (Light4). Для того чтобы вам легче было запомнить, выучите данную фразу:  
**BROG**.  
**B** – Blue (Голубой);  
**R** – Red (Красный);  
**O** – Orange (Оранжевый);  
 **G** – Green (Зеленый).

### Глава 4. Пульт управления рабочей смены

Это специальное оборудование, которое представлено одной кнопкой (переключателем) управление состояния рабочей смены.

|  |  |  |
| --- | --- | --- |
| Параметр | Свойство | Ключ |
| Status | 0 – производство выкл  1 – производство вкл | s |

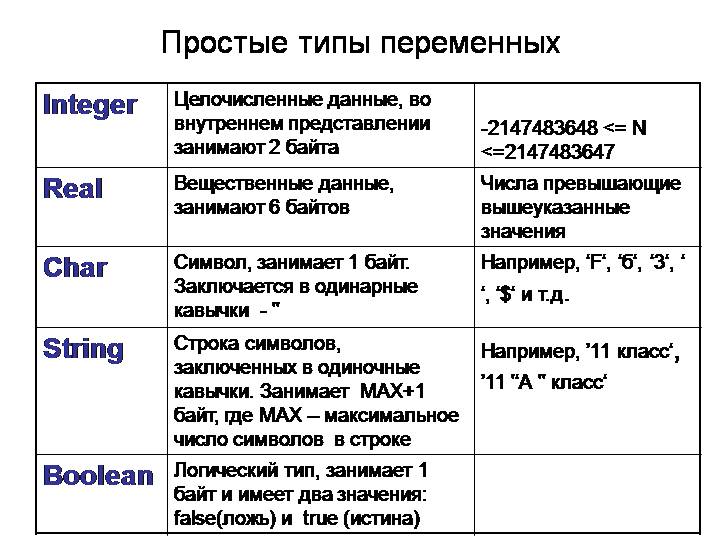
# Часть 4. Параметры, свойства и ключи

### Глава 1. Параметры

Параметрами в платформе «ThingWorx», являются аналогиями переменными в языках программирования. Переменная – это поименованная, либо адресуемая иным способом область памяти, адрес которой можно использовать для осуществления доступа к данным и изменять значение в ходе выполнения программы. Другими словами, переменная – часть отведенной памяти, в которую могут помещаться разные значения (целочисленные, строковые, символы, логические, с плавающей точкой).

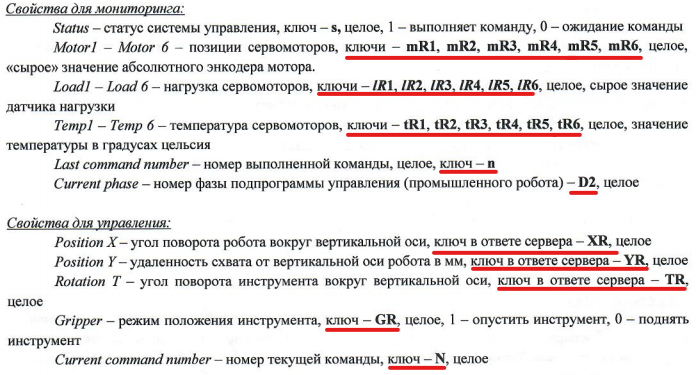
### Глава 2. Свойства

Свойства – это значения, которые задаются пользователем для параметров. Для каждого типа параметров присваиваются строго типизированные значения. Свойства могут быть использованы, как для мониторинга, так и для управления оборудованием, но об этом будет рассказано позже.



### Глава 3. Ключи

Ключи — это специальные переменные заложенные в прошивку оборудования, предназначенные для связывания оборудования с платформой. На каждое свойство оборудования есть свой ключ, с помощью которого и производится подключение. Ключи, как и параметры могут использовать разные типы переменных. В основном ключ – это строковой тип, которые преобразуется в целочисленные через метод “parseInt(“ключ”), но об этом поговорим подробнее позже.



# Часть 5. Оценивание работы

Уровень: «Слабак» (оценка 2) – можете прийти и ничего не делать

Уровень: «Чего-то то стою» (оценка 3) - достаточно будет сделать прием данных и отправление значений на оборудование. Но все управление должно быть реализовано через Mashup иначе задание засчитано не будет.

Уровень: «Личная жизнь не для меня» (оценка 4) – сделать все тоже самое, что и на уровне выше, но также реализовать полуавтоматику, для получения дополнительных баллов. Но этого будет недостаточно, нужно реализовать логирование всех данных, входящих и исходящих, работа с ними (таблицы и графике на Mashup)

Уровень: «Робот» (оценка 5) – сделать все, что написано выше, реализовать автоматизацию и визуализацию данных.

### Пару слов перед изучением

Помните, что вы работаете в команде и делите обязанности, роли между с собой. Не получится добиться эффективной работы, если кто-то один делает больше другого. Общайтесь, кто сделал какую вещь и к чему приступает, чтобы ваш партер знал, может ли он перейти к следующему шагу разработки или продолжить делать другие задачи для получение наивысшего бала.

Помните, что за каждую деталь в визуальном интерфейсе, направленная на получение максимальной информации работы производственной линии – вы получаете баллы, для этого вам нужно создать максимально много параметров и выводить их на экран.

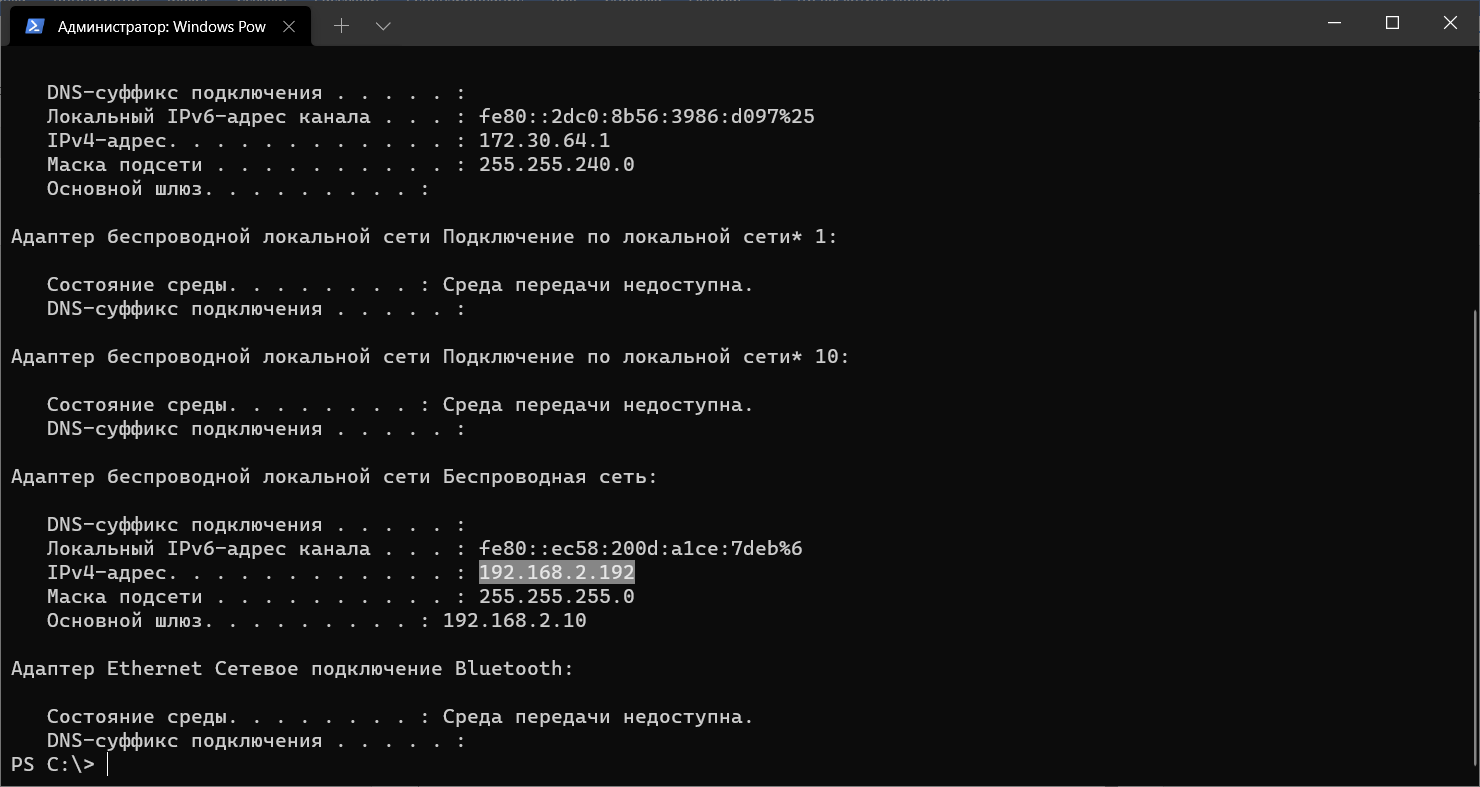
Тем самым мы вас к подводим к главной теме данного абзаца, что пособие может показаться напутанным и не понятным, потому что некоторые моменты делаются чаще и отпускаются, а возвращаемся к ним позже. Для чего это сделано? В угоду оптимизации времени, чтобы получить максимально много первостепенных балов, а после получить второстепенные, так как они имеют вес меньше чем основные.

# Часть 6. Параметры, свойства, Ключи и создание вещей

И так мы разобрались, что такое IoT, облачные технологии, вспомнили (изучили) язык программирования JavaScript и усвоили ООП (Объектное Ориентированное Программирование). Теперь пришло время изучать платформу.

### Глава 1. Авторизация

Чтобы войти на платформу нужно подключиться к серверу, который находится в вашем кабинете. Введите в поисковую строку IP сервера или свой IP (если платформа стоит на вашем ПК). Как узнать IP своего компьютера? Введите в командую строчку: ipconfing и скопируйте пункт (IPv-4 адрес):



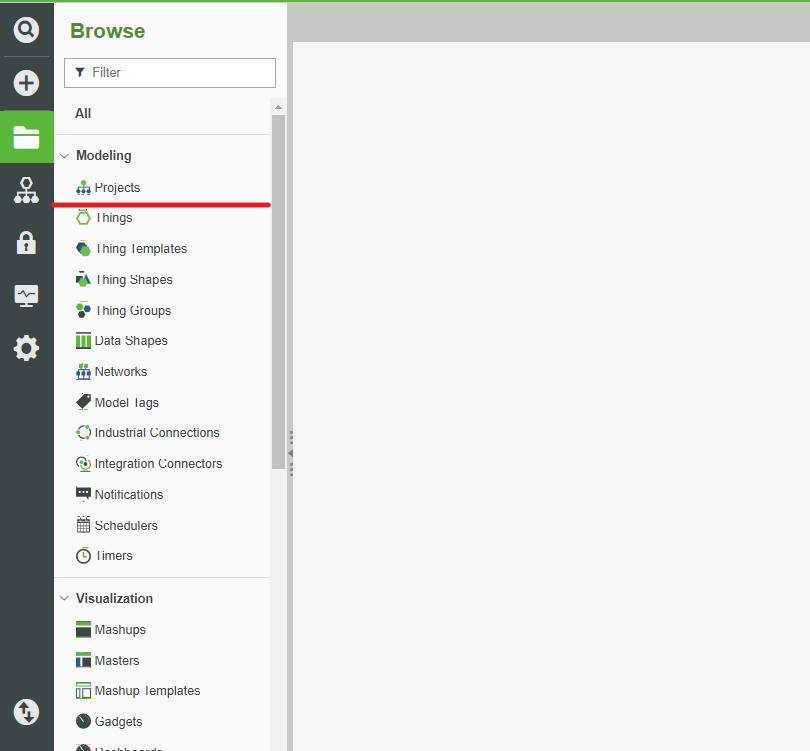
Пароль и логин вы могли устанавливать сами при установке или спросите это у вашего преподавателя.

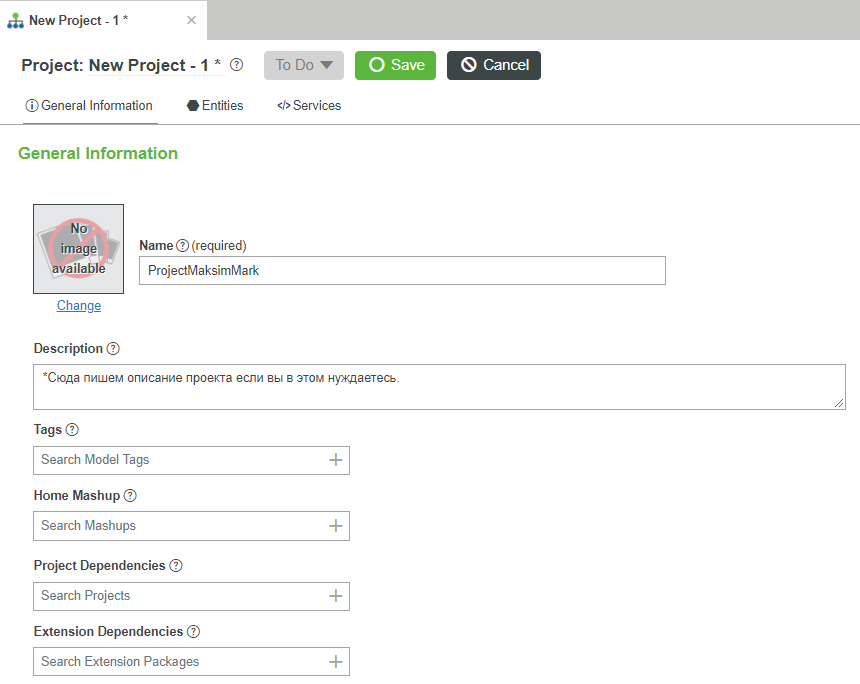
Имя пользователя: **Administrator**  
Пароль: **!QAZxsw2#EDCvfr4**

### Глава 2. Знакомство с элементами. Project

Первый элемент, с которого вы всегда будите начинать – это создание собственного проекта. В нем будут храниться ссылка на все ваши созданные вещи в ходе работы над заданием. То есть это удобный проводник по вашему проекту. Проекты используются с целью организации сущностей в ThingWorx. Сущности, которые собираются в проектах, можно экспортировать и импортировать вместе. Сущность может принадлежать только одному проекту.

Чтобы создать проект нужно нажать на данный элемент (все создаваемые элементы находятся в этой панели):

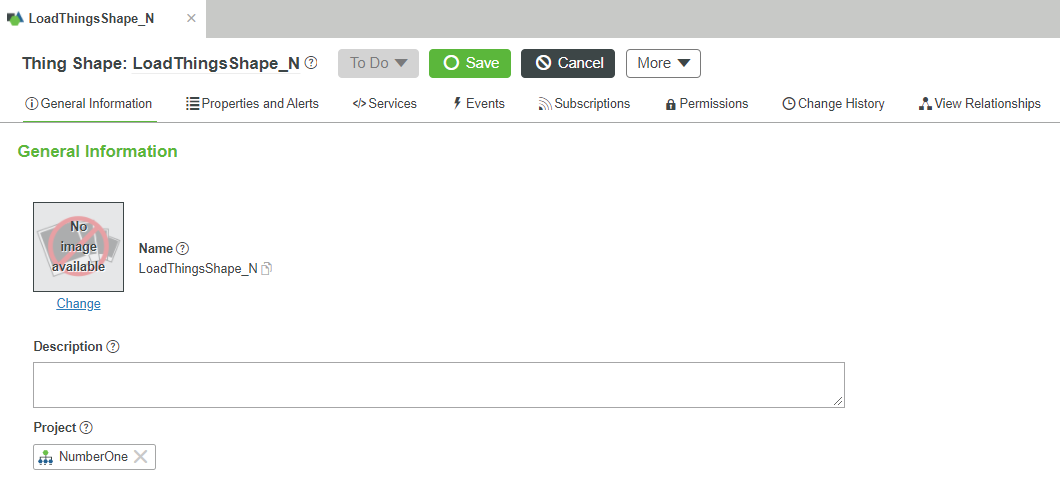
  
  
После у вас откроется окошка создания проекта, где вам нужно ввести имя проекта. Так как вы работаете в парах и для удобства проверки ваших заданий называйте проект так: ProjectИМЯ1ИМЯ2. Пример, в группе работают два человека: Максим и Марк, то название проекта будет называться так – ProjectMaksimMark.

После сохраняем проект и теперь его можно связывать со всеми вашими вещами, главное не забывайте это делать, так как это упростит работу с вашими вещами.

### Глава 3. ThingShapes

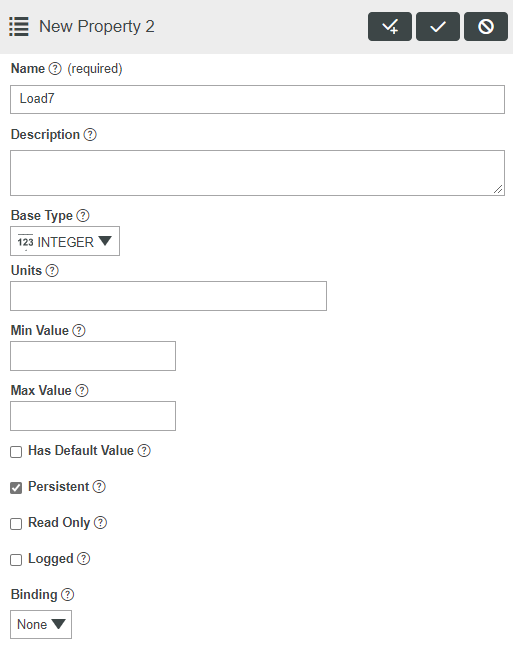
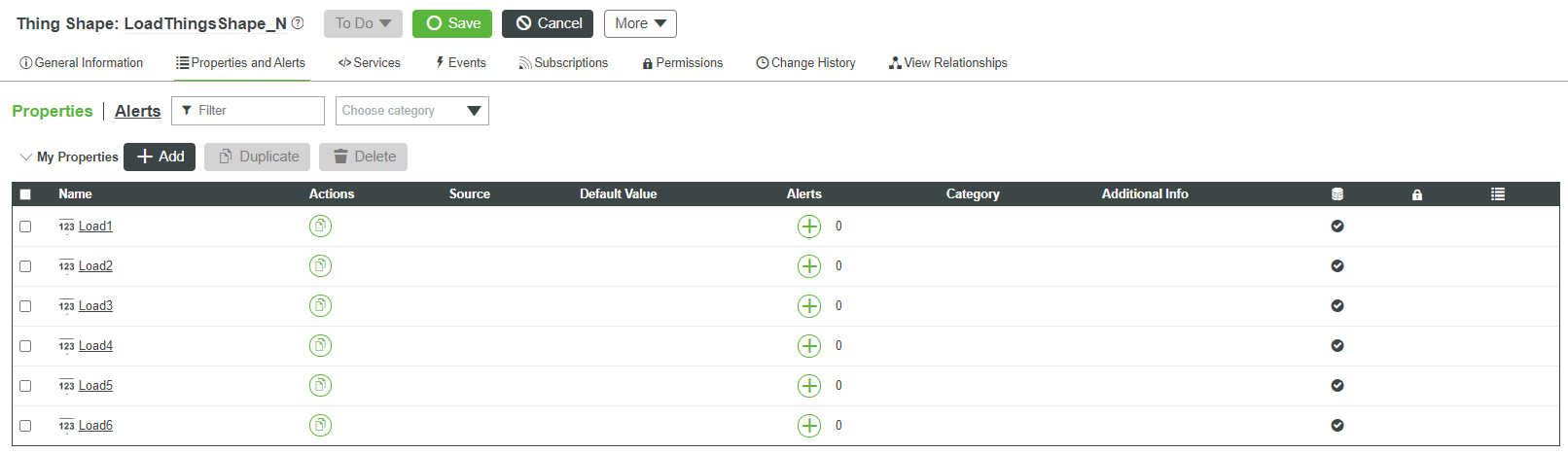
Что это такое? Данная фигура вещей — это противоположность работы ООП. Если вы не знаете, что это такое или как работать с ним, то вверху есть подробное описания ООП, после прочтения вы сможете понять, что к чему. Если все таки вам лень читать, что такое объектное ориентированное программирование, то в двух словах про Thing Shapes – один раз создаются формы вещей с определёнными параметрами такими как Load, Temp, Motor, Light *(не только они, вы можете создавать любые, но для решение нашей задачи нам нужен только такой набор Thing Shapes)* и у наследуются в создании новых Thing (вещей).

Так же выбираем слева Thing Shapes и нажимаем new.



Пример название вещи – LoadThingShapes\_MaskimMark (добавляйте через нижнюю черту имена команды, если этого не делать, то вы зарегистрируете данное имя в системе и другие участники не смогут создать вещь). Так же не забывайте в пункте Project выбрать свой ранее созданный проект.

После того как вы укажите имя вещи и проект переходите во вкладку Properties and Alerts.

Во вкладке Properties and Alerts мы создаем элементы управление / мониторинга.   
Чтобы создать элемент нужно нажать на кнопку «Add». После появится выплывающие окно, в котором мы указываем имя в зависимости от того какой элемент управление мы создаем, будь это Load(нагрузка), Temp(температура), Motor (сервомоторы), Light (светосигнальная лампа). Количество элементов равняется количеству моторов у робота, а значит создаем 6 элементов нагрузки (Load1, Load2 … Load6).  
Также элемент управления ControlThingShapes (PositionX/Y/T, Gripper, Vacuum, Status, CurrentcommandNumber, LastCommandNumber).  
Описание элемента не обязательные и оставляются по собственному желанию.  
Базовый тип элемента за частую мы выбираем целочисленный (int).  
Максимальное и минимальное значение не вводятся в свойства элемента, а реализуются с помощью сервисов, о которых мы поговорим позже. И почти во всех, за исключение некоторых условия, выставляется галочка на пункт Persistent. Данная галочка предназначена для сохранения последнего значения. Чтобы узнать сколько нужно создавать элементов управления, прочитайте документацию к вашему оборудованию. По этому же принципу создаются ThingShpes для Temp, Motor и Light.

### Глава 4. DataShapeS, что это такое и с чем едят?

Data Shapes представляют данные в вашей модели. Форма данных – это именованный набор определений полей и связанных метаданных, проще говоря база данных. Каждое поле в Data Shapes имеет тип данных (строковое, целочисленное, булевые значения и т д). ThingWorx имеет определенный набор базовых типов.

Data Shapes помогают создать приложения, приложения принимают данные в определенную базу данных, приложение знает о том, как нужно формировать данные из БД. Например, если вы переместите данные в таблицу в визуальном интерфейсе, таблица знает, как нужно их отобразить, благодаря ранее созданной таблицей в БД. Таблица уже знает, какие поля являются числами, строками или датами. Когда вы настраиваете, как сетка должна отображать данные \ принимать данные, если вы знаете какие данные будут поступать в таблицу – это значительно упростит настройку.

Для хранения данных создаются определенные вещи: потоки, потоки создания ценности и таблицы данных. Когда вы определяете вещь для хранения данных, вы должны определить форму данных (кроме потоков значений).

Data Shapes используются во многих случаях, а не просто как определения для потоков, потоков значений и таблиц данных. Thing Shapes также используются, когда вам нужно описать набор данных. Например, когда вы определяете информационный вывод для реализации службы, вы используете Data Shapes для описания набора результатов вывода. У вас может быть параметр Thing типа infotable(таблица), а также можно указать форму данных, которая описывает это свойство.

Проще говоря, мы создаем шаблон базы данных с определёнными базовыми типами, в которые будут записываться значения и читать их оттуда из таблицы. Data Shapes позволит нам создать полностью автоматизированную сборку, когда будет приходить код сборки на производственную линию.

Создание DataShapes

Чтобы создать DataShapes, нужно выбрать его в боковом меню. Мы будем создавать дата шейпы для хранения в них коды сборок, координаты для перемещения, но это не все, дополнительные Data Shapes вы будите создавать в главе логирование, которое будет позже.

### Глава 5. Things (Создание робота)

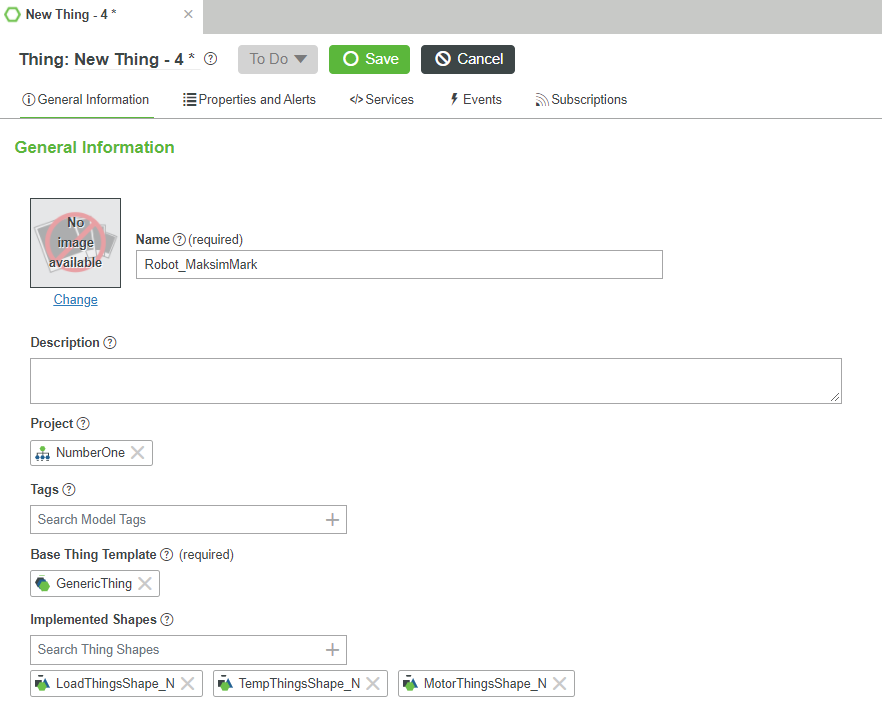
Вещи являются представлениями физических устройств, изделий, систем, людей или процессов, которые имеют свойства и бизнес-логику. Все вещи основываются на шаблонах вещи (наследование) и могут реализовать один или несколько профилей вещи (состав). Рекомендуется создать шаблон вещи, чтобы описать вещь, а затем создать экземпляр этого шаблона вещи как вещь. Такая практика использует наследование в модели и сокращает время, затрачиваемое на поддержание и обновление модели.

Вещь может иметь свои собственные свойства, сервисы, события и подписки и может наследовать другие свойства, сервисы, события и подписки от ее шаблона вещи и профилей вещи. Способ моделирования взаимосвязанных вещей, шаблонов вещи и профилей вещи является ключевым, чтобы создать решение простым для развертывания и поддерживания его в будущем при изменении физических активов. У конечных пользователей появится интерфейс с вещами для получения информации в приложениях и для чтения и записи данных.

Удаленные от ThingWorx устройство или источник данных, к которым обращаются по сети (интрасеть/Интернет/WAN/LAN), представляются удаленной вещью.

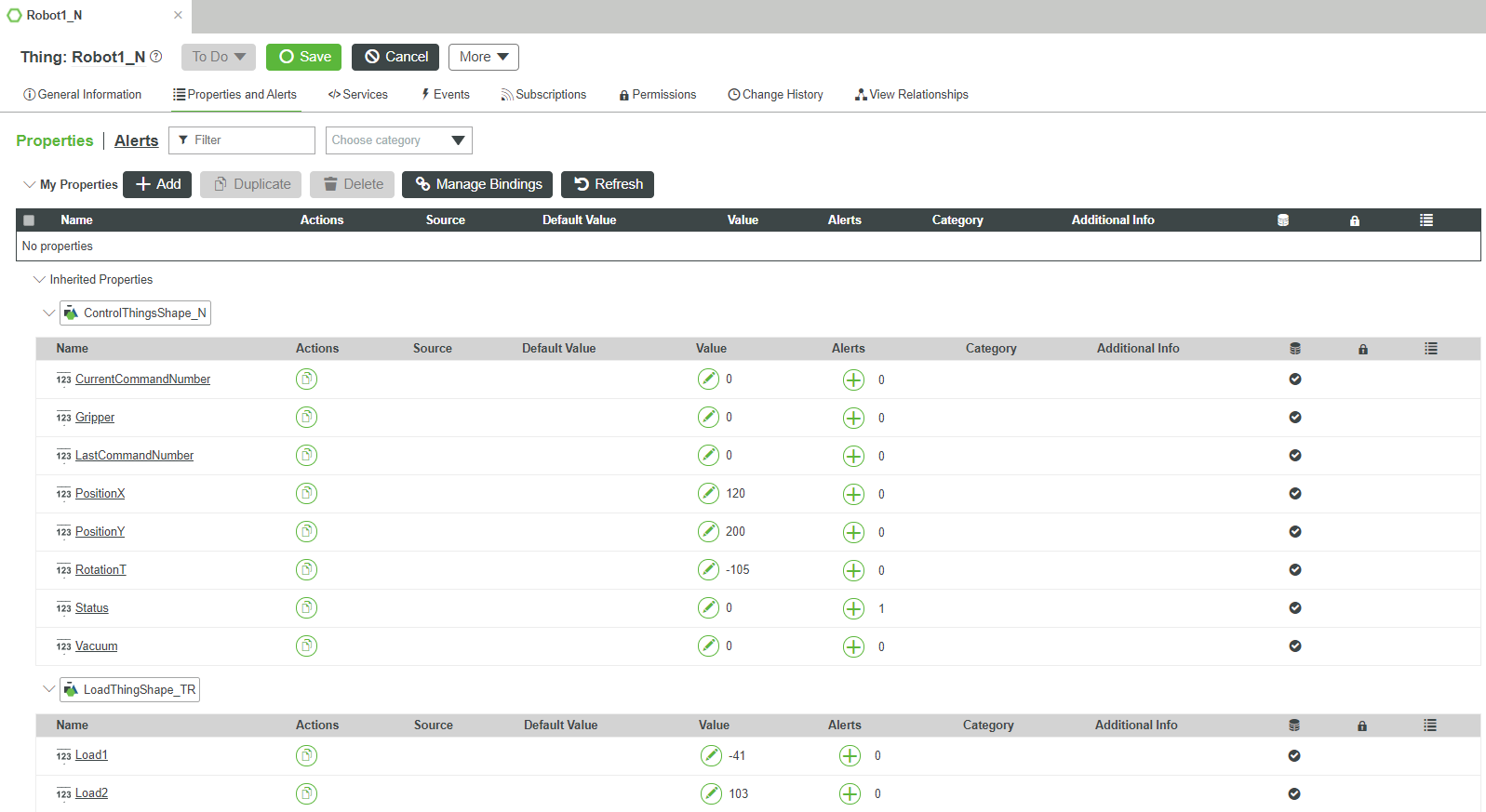
Из вышеперечисленного описание можно понять, что Thing нужна для реализации дублирования физической модели в виртуальную с параметрами управления, но не только для этого, а также создания своих элементов управления и связать вещи между с собой для синхронной работы.

Чтобы создать вещь, нужно перейти во вкладку Thing и нажать «new».  
В открой вкладке нам нужно указать имя вещи, НЕ ЗАБЫВАЕМ наше правило установлении имени. Пример: Robot\_MaksimMark.



В пункте «Project» выбираем свой ранее созданный проект.  
В пункте «Base Thing Templete» выбираем GenericThing, это нужно для реализации базовых инструкций вещи, таких как GetProperties / SetProperties. Кто имеет опыт программирования с Си подобными языками программирования, то могут сравнить это с подключением базовых библиотек языка программирования. Для тех, кто никогда не кодил на подобных языках программирования, но прочитали главу про ООП, то для вас можно объяснить это так – мы унаследуем класс GenericThing и принимает оттуда свойства, параметры и т. д.  
В пункте Implemented Shapes мы указываем ранее созданные Thing Shapes (MotorThingShapes, TempThingShapes, LoadThingShapes, ControlThingShapes).

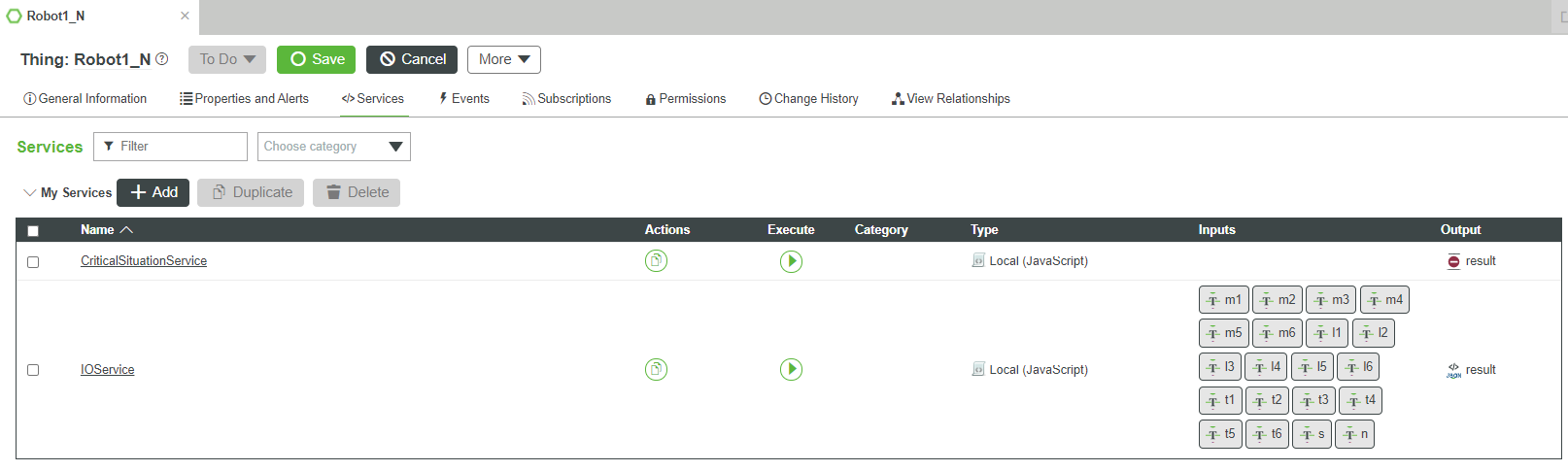
Если вы откроете вкладку Properties, то увидите, что там уже есть эелементы управления, которые мы создали с помощью Thing Shapes и добавили во вкладе Implemented Shapes. В нашем случае у нас 3 робота и при каждом его создании нам бы приходилось добавлять все эти элементы заново, но за счет ранее созданных Thing Shapes мы облегчили себе работу сделав это всего один раз.



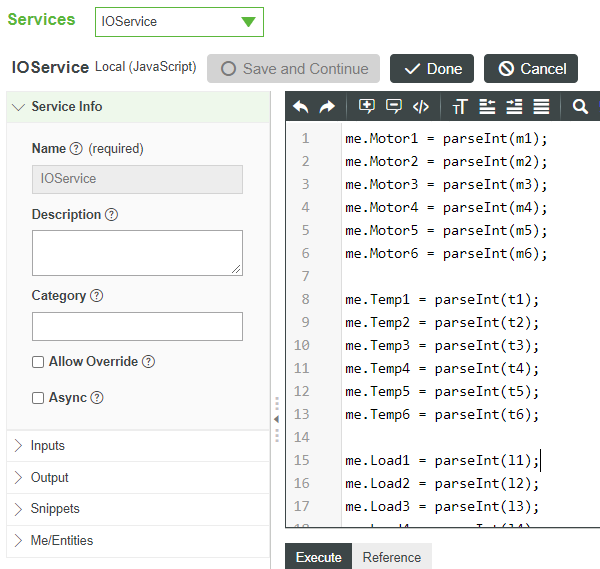
**Сервисы вещи**

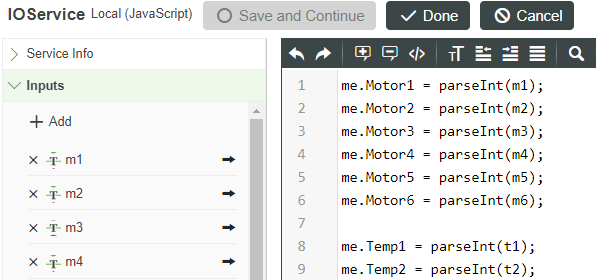
Сервисы вещи являются функциями, которые может выполнить вещь. Каждая вещь может иметь один или несколько сервисов. Можно определить сервисы в профиле вещи, в шаблоне вещи или на уровне вещи. Простым примером сервиса является запрос, записанный для вещи базы данных.

Теперь во вкладке Service мы создаем сервис для обращение к вещи и принятие от неё данных.



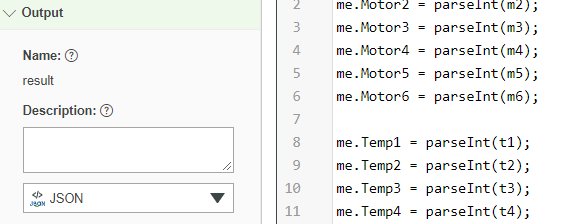
Чтобы создать свой сервис нажимаем на кнопку «Add», после у нас открывается новое окно.

В данном окне нам нужно указать имя сервиса, ключи для мониторинга данных и формат вывода скрипта. Стандартные сервисы для управления вещью рекомендовано называть «IOService».  
После во вкладке «Inputs» нужно добавить ключи, которые прописанные в инструкции физической вещи (Робот, сигнальная лампа, удаленный терминал, режим смены).



Нажав на кнопку «Add» появится окно добавления:  


В имени пишем имя ключа, а в «Base Type» указываем тип входных данных строковой, так как все данные с оборудования приходят в строковых значениях, которые нужно будет преобразовывать через метод parseInt(). Ключи добавляются только в те вещи, которые обмениваются данными с физическими вещами.

Во вкладке «Output» нужно выбрать формат обмена данными – JSON. Данный формат ставиться только в те вещи, которые обмениваются данными с физическими вещами.  


И последнее это написание логики работы вещи (скрипт). В большом окошке справа прописывается весь скрипт.  


Чтобы обратиться к параметрам во вкладке «Properties» нужно написать префикс «me». Например, чтобы обратиться к Motor1 нужно написать именно так – «me.Motor1».

Теперь перейдем к написанию скрипта взаимодействия платформы с вещью, на данный момент мы опишем скрипт для робота (другие вещи такие как удаленный терминал, светосигнальная лампа, управление рабочей смены будут описаны ниже).

Параметры для мониторинга мы не можем изменять, а только читать с робота, к таким данным относятся параметры:

1. Motor1 – Motor6;

2. Temp1 – Temp6;

3. Load1 – Load6;

4. LastCommandNuber;

5. Status.

Параметры для управления мы можем изменять, к таким данным относятся параметры:

1. Position X;

2. Position Y;

3. Rotation T;

4. Gripper;

5. Vacuum;

6.CurrentCommandNumber.

У управляемых параметрах есть ключи, но они задаются они другим способом, по стандарту JSON формату. В предоставленном коде вы сможете увидеть, как они реализованы. Ниже представлен весь код скрипта для взаимодействия с роботом.

me.Motor1 = parseInt(m1);

me.Motor2 = parseInt(m2);

me.Motor3 = parseInt(m3);

me.Motor4 = parseInt(m4);

me.Motor5 = parseInt(m5);

me.Motor6 = parseInt(m6);

me.Temp1 = parseInt(t1);

me.Temp2 = parseInt(t2);

me.Temp3 = parseInt(t3);

me.Temp4 = parseInt(t4);

me.Temp5 = parseInt(t5);

me.Temp6 = parseInt(t6);

me.Load1 = parseInt(l1);

me.Load2 = parseInt(l2);

me.Load3 = parseInt(l3);

me.Load4 = parseInt(l4);

me.Load5 = parseInt(l5);

me.Load6 = parseInt(l6);

me.Status = parseInt(s);

me.CurrentCommandNumber = parseInt(n);

result = {

    "X": me.PositionX,

    "Y": me.PositionY,

    "T": me.RotationT,

    "G": me.Gripper,

    "V": me.Vacuum,

    "N": me.LastCommandNumber

};

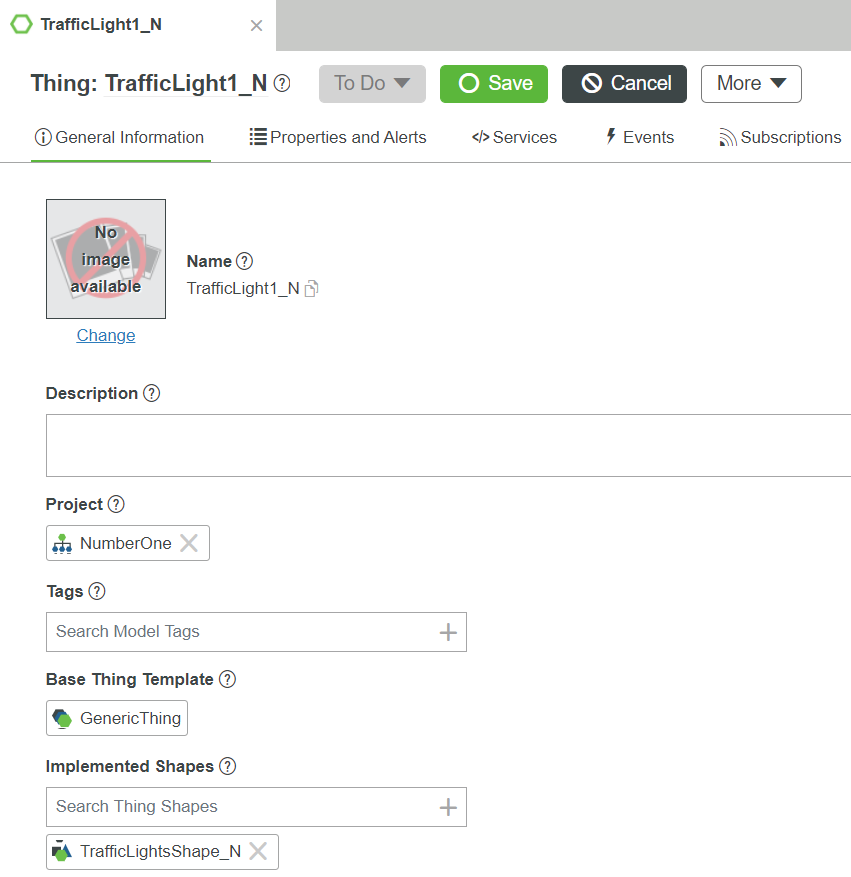
Параметры, которые находятся до result являются мониторингом и только принимают значения, но не отправляют их обратно на робота.

Параметры управления находятся в result и собираются в пакет данных, которые принимают значения и отправляются на робота. Ключи управляемых параметров прописываются через двойные кавычки.   
Пример вы можете увидеть в коде выше.

Теперь если вы запустить робота через утилиту IoT на вашем сервере, то сможете управлять им изменяя параметры во вкладке «Properties». Повторяю, не пробуйте изменять параметры мониторинга они все равно не отправляются на робота, а также не вводите критические значения робота, чтобы их узнать спросите куратора вашей компетенции.

### Глава 6. Cоздание светосигнальной ЛАМПЫ (TrafficLights)

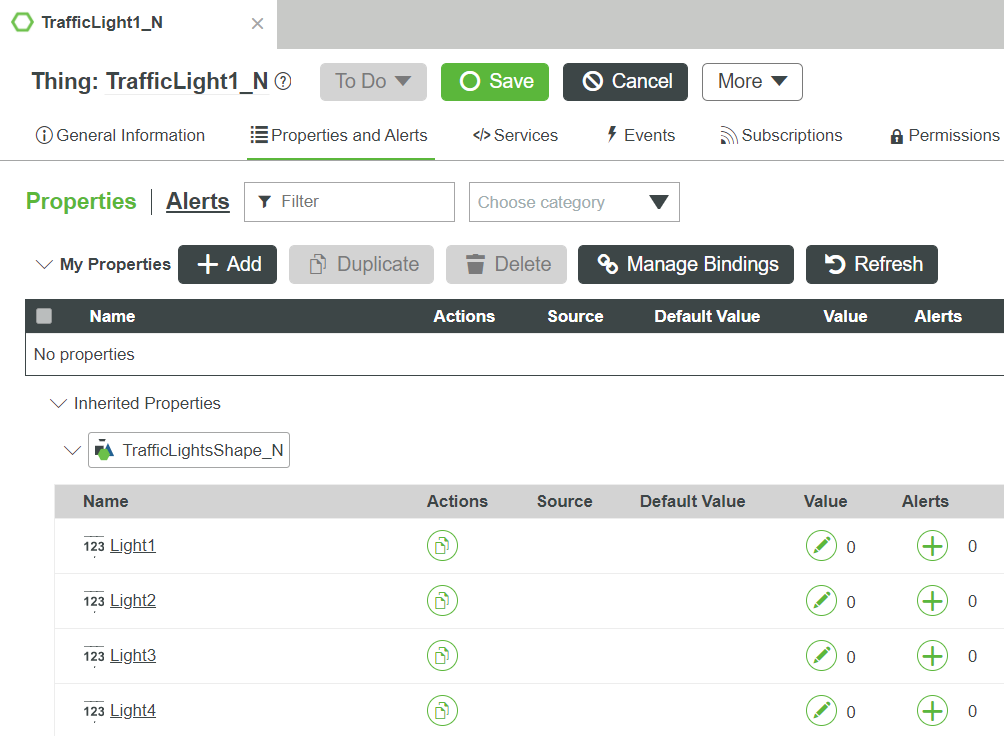
В «Часть 4. Things (Создание робота)» вы научились создавать робота в платформе, давайте разберем как продублировать физическую модель лампы в виртуальную с параметрами для мониторинга и управления. Инструкция будет аналогичная, даже проще, потому что тут меньше значений для управления \ мониторинга данных.

Мы так же создаем новую вещь «Things» и даем ей имя – TrafficLight1\_MaksimMark (светосигнальных ламп столько же сколько и роботов). Добавляем в свой проект (Project).  


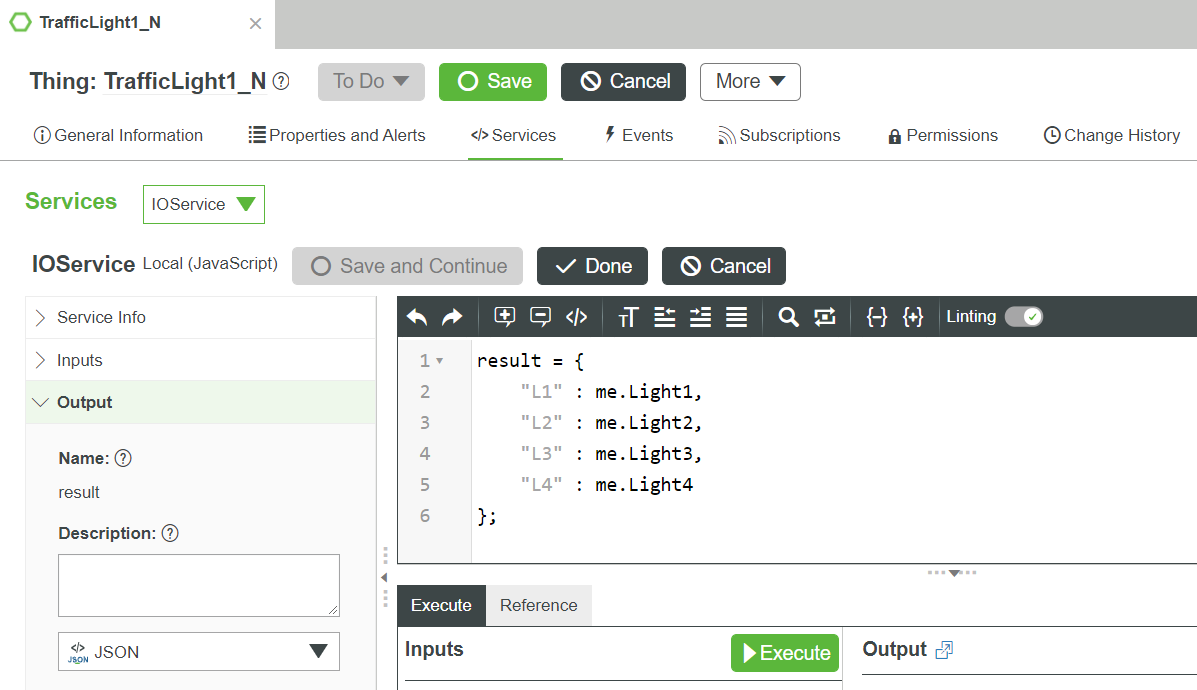
Описание пишем по желанию, если оно вам нужно (равносильно комментированию в программном коде).

В «Base Thing Templete» выбираем – Generic Thing.

В последнем пункте выбираем ранее созданный Thing Shapes для нашего TrafficLight.

Во вкладке Properties вы заметите, что тут появятся параметры, ранее созданные в Thing Shapes.  


Во вкладке Service создаем сервис с именем – IOService.  
Во вкладке Output выставляем вывод в формате – JSON.  
Ключи в этом случае не добавляем, так как они задаются в result через двойные кавычки, параметры светосигнальной лампы являются, как мониторингом, так и управлением.



Полный код реализации сервиса:

result = {

    "L1": me.Light1,

    "L2": me.Light2,

    "L3": me.Light3,

    "L4": me.Light4

};

### Глава 7. Создание удаленного терминала (RemoteTerminal)

Создаем новую вещь «Things» и даем ей имя – RemoteTerminal\_MaksimMark. Не забываем добавить его в свой проект.

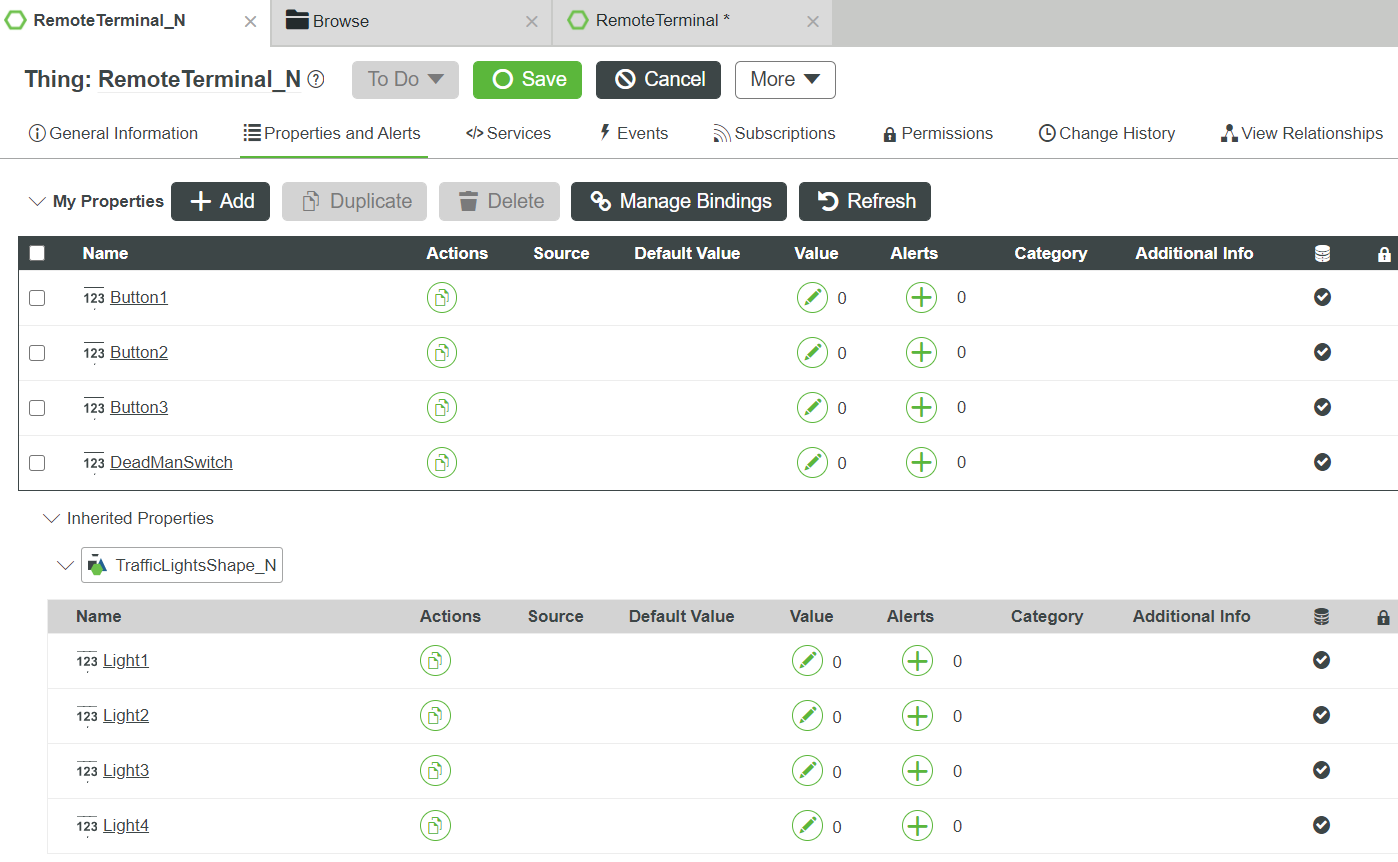


В «Base Thing Templete» выбираем – GenericThing (его нужно выставлять каждый раз при создании новой вещи, других базовых шаблонов мы использовать не будем).

Добавляем в «Implemented Shapes» форму вещи для светосигнальной лампы, так как на удаленном терминале присутствует дублирование индикации TraffiLight (светосигнальной лампы).

Во вкладе Properties мы увидим параметры Light1-4, но нам нужно создать еще несколько параметров – это:

1. DeadManSwitch – кнопка аварийной остановки  
   Name - DeadManSwitch;  
   Base Type – INTEGER;  
   Persistent (сохранение последнего значения) – true.
2. Button [1-3] (три кнопки) – запуск автоматической сборки, пауза, стоп.  
   Name - Button [1-3];  
   Base Type – INTEGER;  
   Persistent (сохранение последнего значения) – true.



Во вкладке Service создаем сервис с именем – IOService.  
В Inputs создаем 4 ключа:

1. p – string;
2. b1 – string;
3. b2 – string;
4. b3 – string.

В Output ставим формат вывода – JSON.

Полный код реализации сервиса:

me.DeadManSwitch = parseInt(p);

me.Button1 = parseInt(b1);

me.Button2 = parseInt(b2);

me.Button3 = parseInt(b3);

result = {

    "L1": me.Light1,

    "L2": me.Light2,

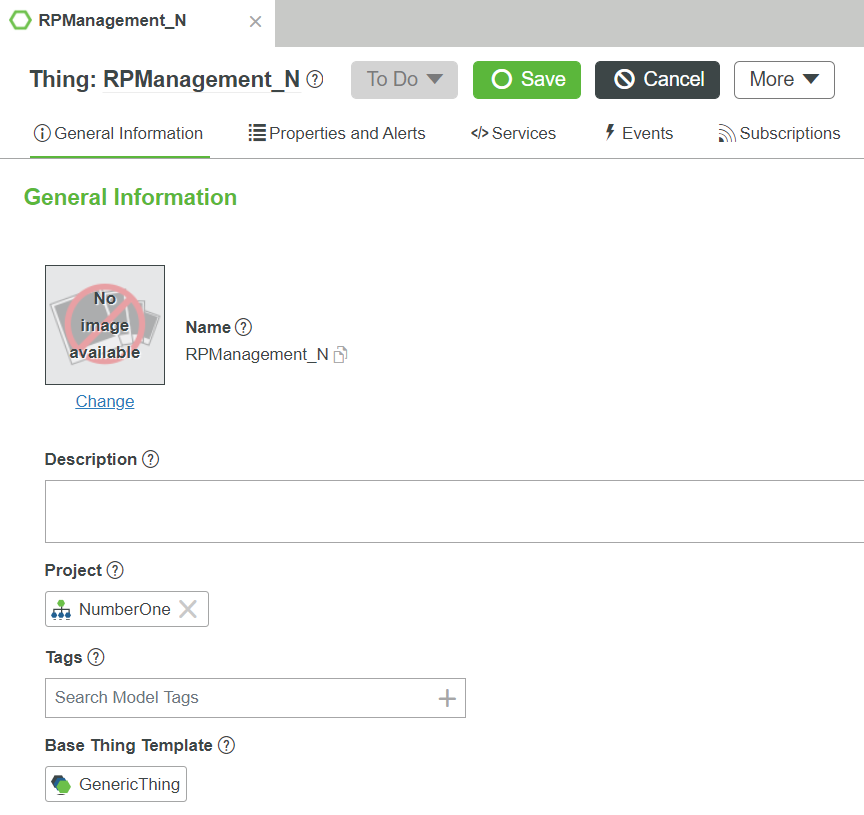
    "L3": me.Light3,

    "L4": me.Light4

};

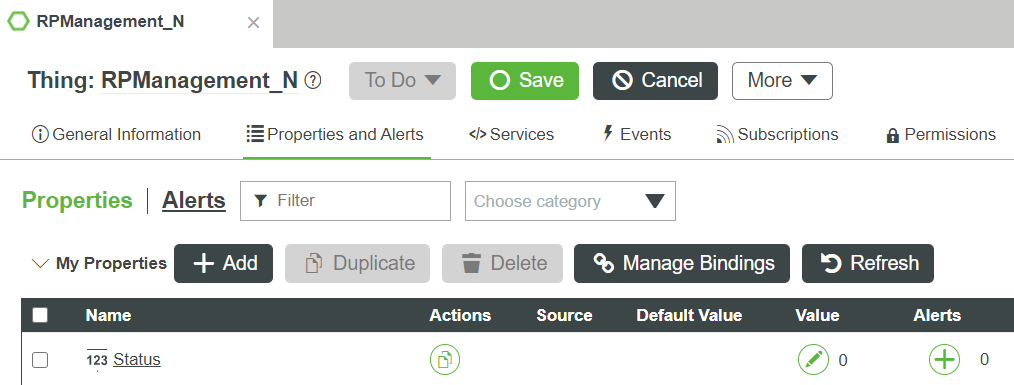
### Глава 8. Создание управление производственной ячейки (RPMAnagment)

Создаем новую вещь «Things» и даем ей имя – RPManagment\_MaksimMark. Не забываем добавить его в свой проект. Не забываем добавить его в свой проект. Ну и как обычно – GenericThing. Так как в данной вещи используется только одна кнопка, для нее не нужны Thing Shapes.



В Properties создаем один параметр:

1. Name – Status (показывает запущенно производство или нет)  
   Base Type – INTEGER;  
   Persistent – false (выставляем данное значение на false, чтобы не запоминался статус работы производства при вкл. ThingWorx, а то роботы сразу начнут выполнять последние действия).



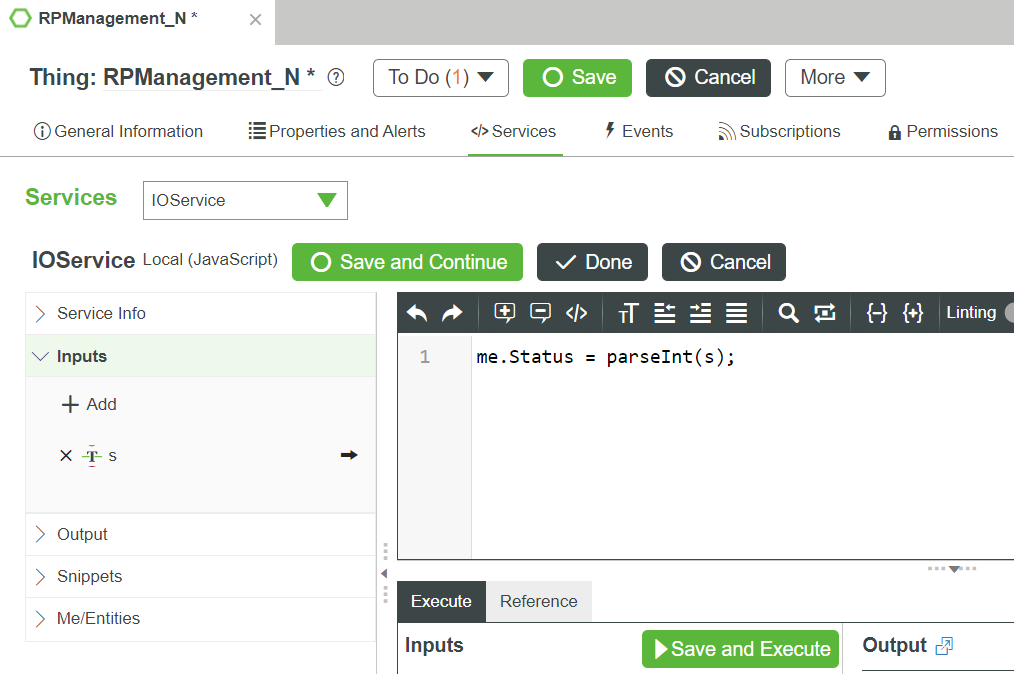
Во вкладке Service создаем сервис с именем – IOService.  
В Inputs создаем 1 ключ:

1. s – string;

В Output ставим формат вывода – NOTHING.

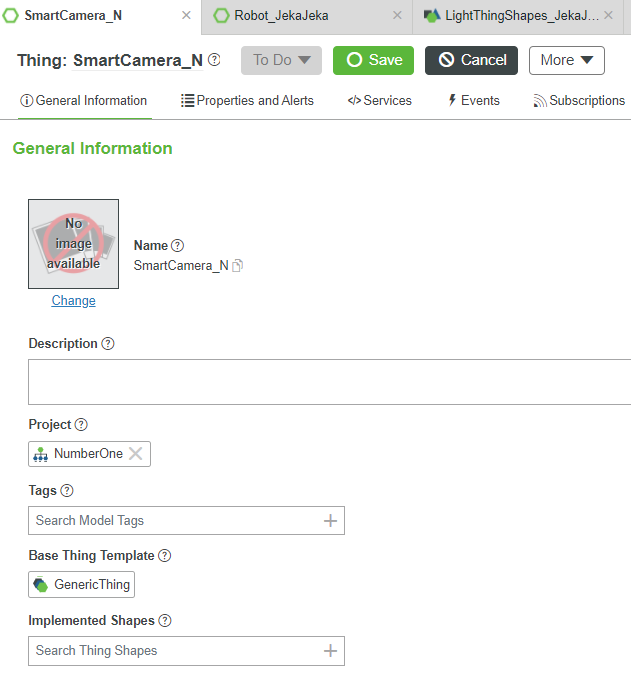
Полный код реализации сервиса:

me.Status = parseInt(s);



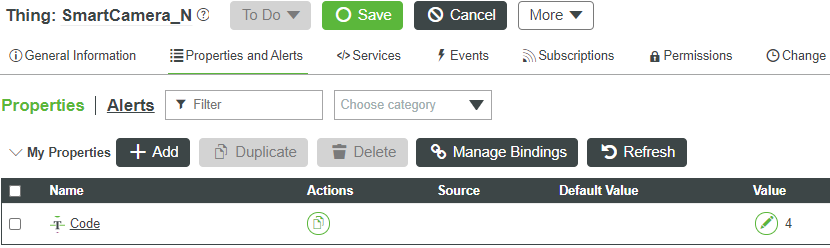
### Глава 9. Умная камера (SmartCamera)

Создаем новую вещь «Things» и даем ей имя – SmartCamera\_MaksimMark. Не забываем добавить его в свой проект. Не забываем добавить его в свой проект. Ну и как обычно – GenericThing. Так как в данной вещи используется только один параметр для принятие входных данных, для нее не нужны Thing Shapes.



В Properties создаем один параметр:

1. Name – Code (показывает запущенно производство или нет)  
   Base Type – String;  
   Persistent – True.



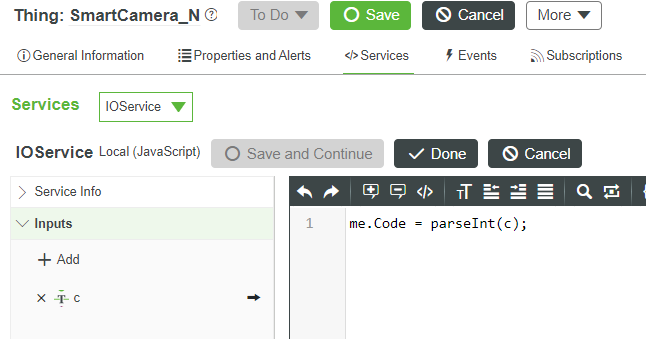
Во вкладке Service создаем сервис с именем – IOService.  
В Inputs создаем 1 ключ:

* 1. c – string;

В Output ставим формат вывода – NOTHING.

Полный код реализации сервиса:

me.Code = parseInt(c);



### Глава 10. Подписки (Subscription)

Подписки являются сервисами, которые получают события и реагируют на них. Подписка содержит источник, обычно это вещь. Вещь может иметь подписку на событие, на которое отвечает действием. Например, если сущность инициирует событие Двигатель перегревается, она может подписаться на это событие, запуская подписку выключить двигатель. Вещи могут наследовать подписки от шаблонов вещи и профилей вещи, которые они используют.

Подписка подобна стандартному сервису, но она связана с событием явно. Это позволяет отсоединить событие от кода, который отвечает на него. Чтобы реагировать на событие, можно, как и для сервиса, применить пользовательскую бизнес-логику. Можно использовать возможности модели, отправляя электронные письма через вещь почтового сервера, записывая в базу данных или вызывая любые сервисы, доступные в платформе. Подписка, в отличие от сервиса, не имеет явно возвращаемых выходных данных, но может вызывать любой другой сервис в модели, к которому имеет доступ контекст безопасности потока. Контекст безопасности потока для подписки задается, как тот же контекст безопасности потока инициированного события. Можно использовать ту же среду редактирования JavaScript, которая используется для реализации сервиса.

Подписки имеют определенные входные данные, которые являются пакетом данных, выпущенным событием; их называют данными события. Если сущность подписана на определенное событие, данные этого события передаются в функцию подписки. Данные события описываются в структуре данных события. В реализации подписки данные, которые передаются из события, действуют как входные данные для функции сценария. Например, если объект подписан на событие изменения данных свойства вещи, то вызывается функция сценария подписки, и значение свойства вещи вместе с другими соответствующими данными из события передается в функцию как часть данных события.

Несколько сущностей может подписаться на одно и то же событие. Каждая сущность получает вызов подписки с переданными данными события. Сущность может выполнить любое действие из сценария подписки, которое требуется для выполнения требований к решению.

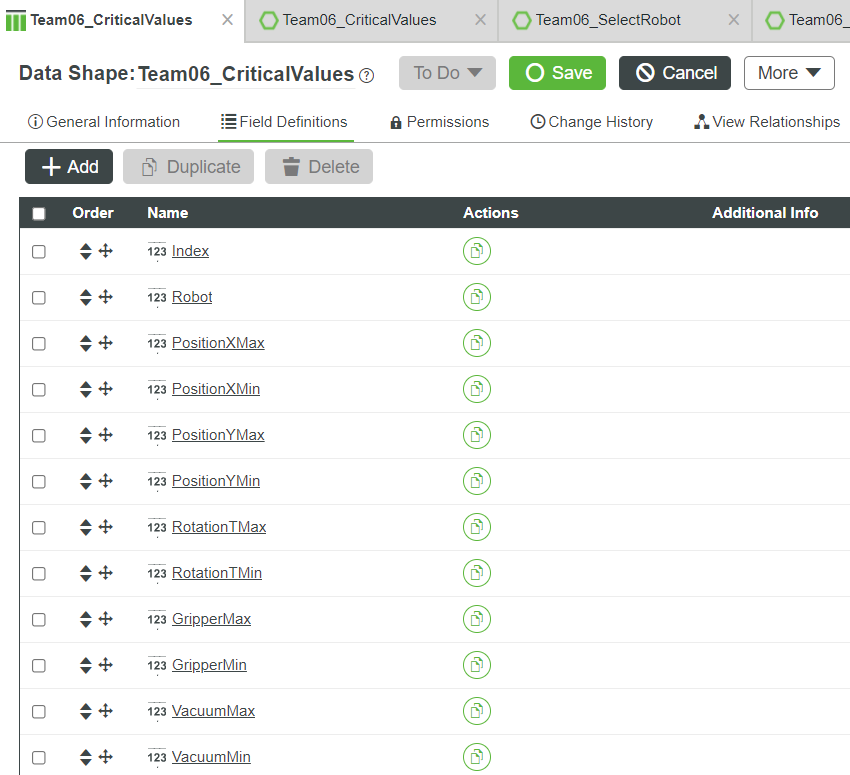
Преимущества этого метода над использованием сервиса, который вызывается из другого сервиса, состоят в том, что одно событие может использоваться для нескольких подписок, а события вызываются на основе активности системы и вмешательство пользователя не требуется. Если на одно событие должны подписаться несколько вещей, событие можно использовать вместо цепочки нескольких сервисов.

### Глава 11. Критические значения (Critical Values)

После создания оборудования, мы должны создать таблицу с критическими значениями для оборудования, которые мы сможем изменить в любой момент через Mashup. В прошлом документе мы делали кучу параметров, теперь тут будет одна таблица, которую мы сможешь заполнять и изменять через Mushup, но сокращение параметров привело к созданию более сложных сервисов. Решать вам каким способом будите пользоваться.

DataShape

Чтобы заполнить таблицу, сначала нужно сделать ее шаблон. Для этого мы делаем DataShape.



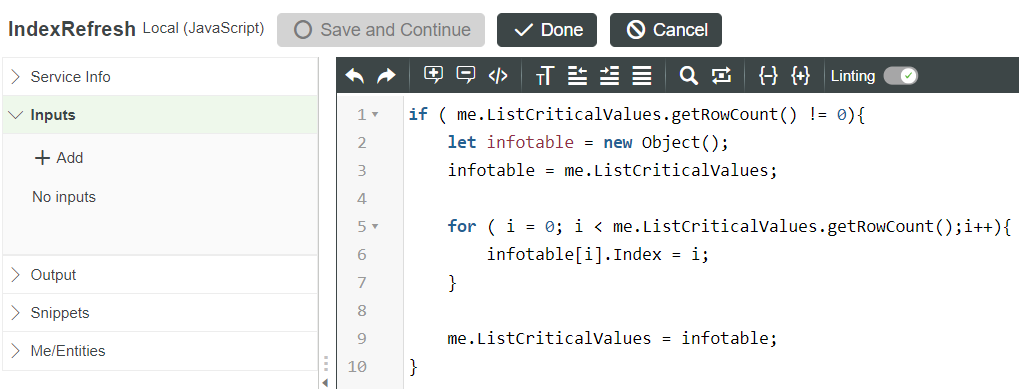
Из нового сюда нужно добавить индекс, робота (чтобы понимать к какому роботу относится строчка в таблице) и PositionMax и PositionMin, дальше перечислить все моторы, температуры и нагрузки.

После создается вещь с именем «CriticalValues» и добавляется одни параметр – infotable + Persistent + DataShape(который мы сделали ранее). Можем сразу заполнить его как нам нужно.

Service

Теперь нужно сделать сервисы для работы с таблицей, добавление, удаление и изменение.

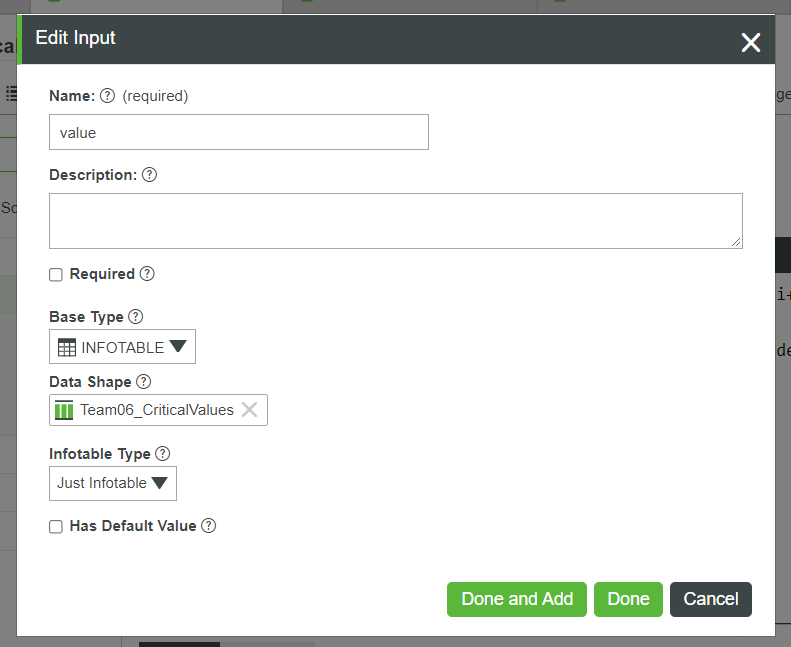
«IndexRefresh» (обновление значений Index, после удаления или добавление строчки)



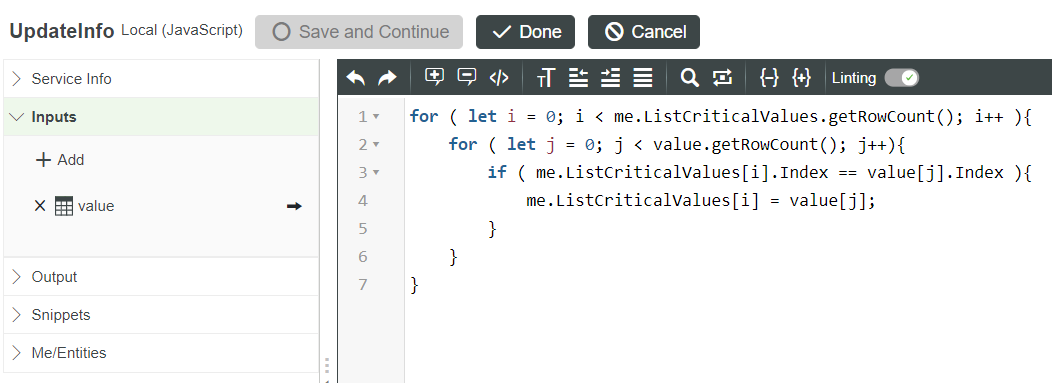
«UpdateInfo» (обновление значений в таблице, если оно было изменено через Mashup)

В это сервисе создается «Inputs» - входные данные. Они нам будут приходить из Mashup, когда пользователь изменит значение в таблице и нажмет сохранить.

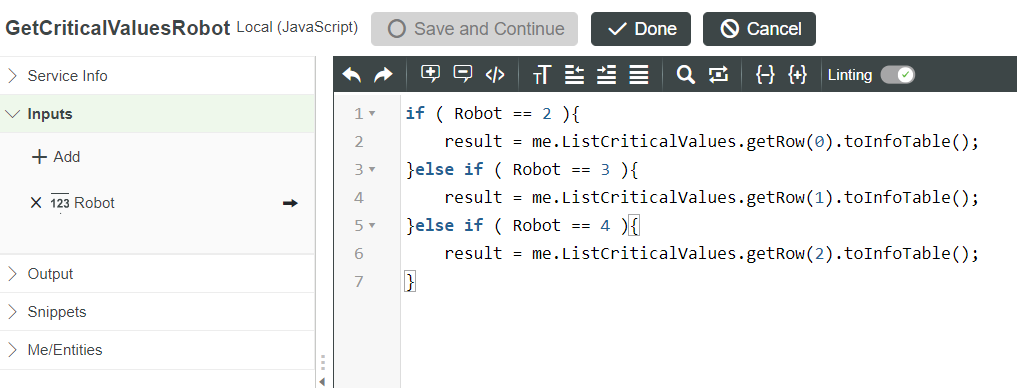
Во вкладке Inputs добавляем значение типа – Infotable с DataShape, который мы делали под «CriticalValues», чтобы данные которые приходили с Mashup, были такого же формата, как и в таблице.



После добавляем такой код:

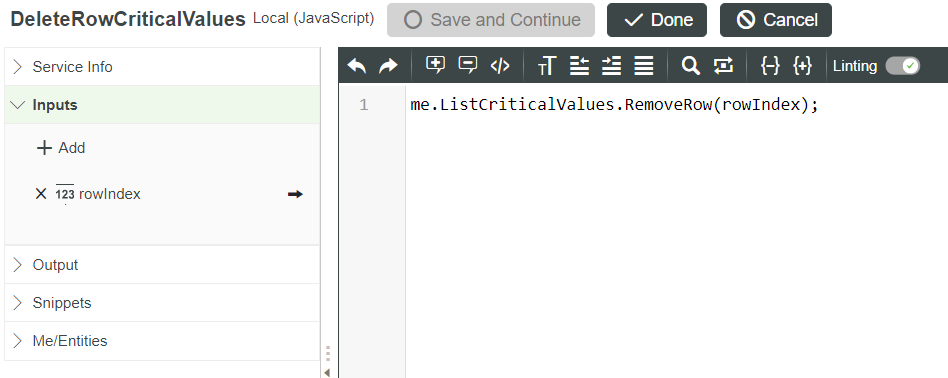


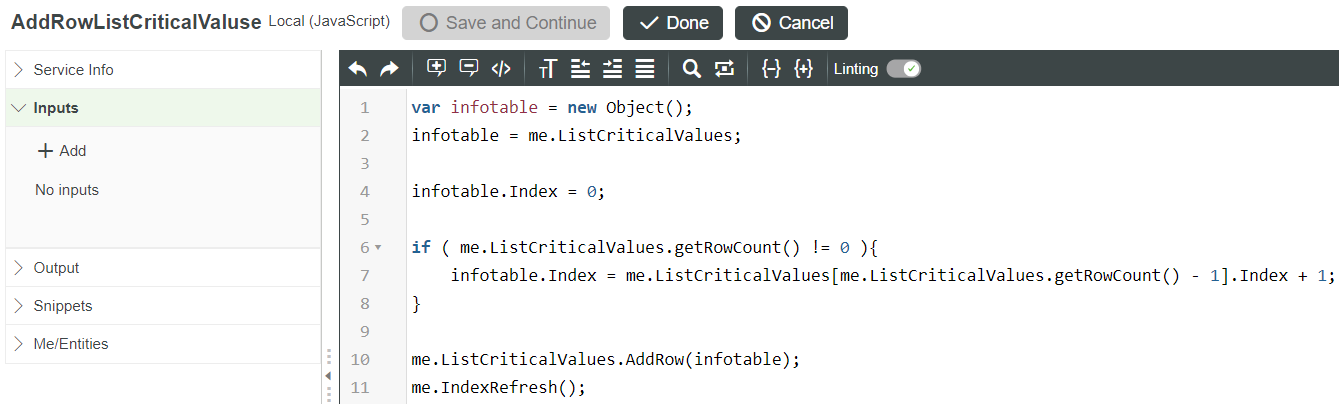
«GetCriticalValuesRobot» (получить критические значение выбранного робота)



«DeleteAllRowCriticalValues» (удалить все поля в таблице) 

«DeleteRowCriticalValues» (удалить выбранную строчку в табоице)

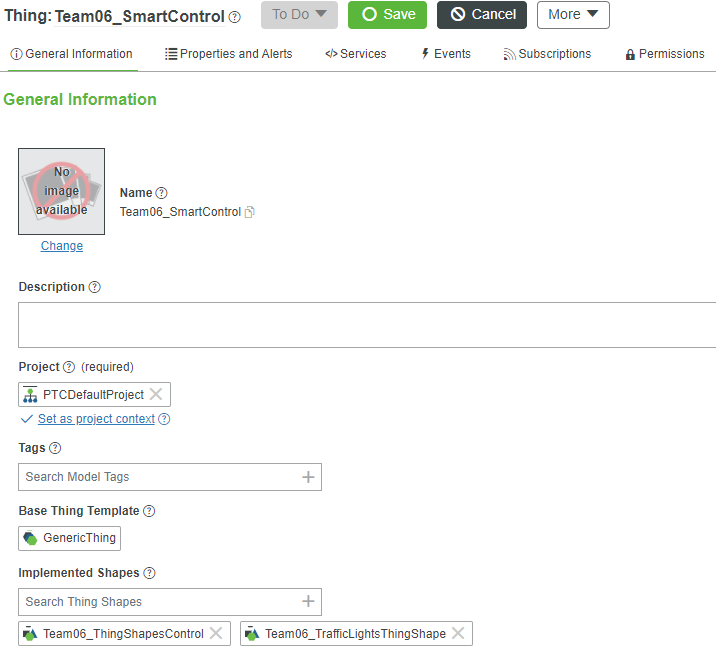


«AddRowListCriticalValuse» (добавление новой строчки в таблицу):  


Остальные действия производится в Mashup, изучение которого нужно проходить в техникуме :)

### Глава 12. Управления (Ручное, полу-автоматическое, автоматическое)

В более ранний версии документа мы создавали вещь под именем «Auto», теперь изменилось не только название вещи на «SmartControl», но и ее составляющая, теперь она стала вмещать в себя «ManualControl» и «SemiAuto» - данное решение было принято для удобства реализации программного кода для управления оборудованием.



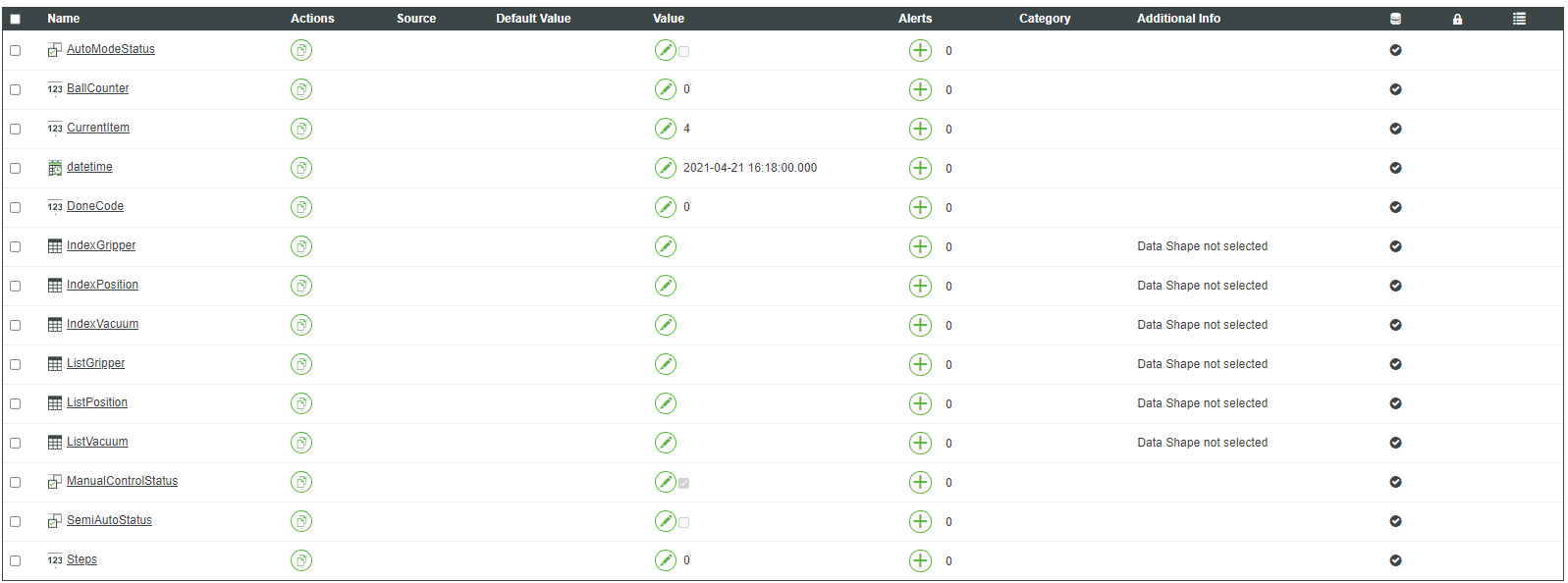
В данную вещь мы добавляем ThingShapes управления и светосигнального светофора.

Параметры данной вещи будут зависеть от задания, которое нужно выполнить, сервисы будут такими же, но изменяться будут также в зависимости от задания. Давайте начнем разбираться с данной вещью.

Properties and Alerts

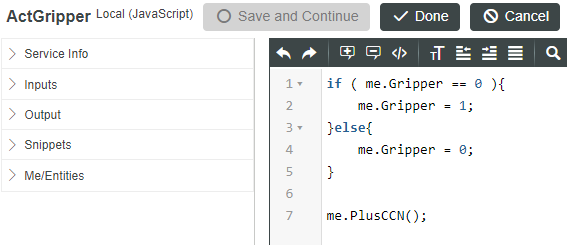
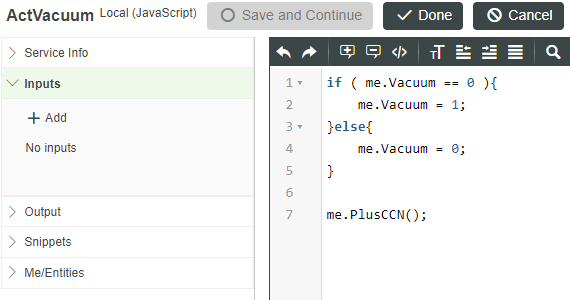
1. AutoModeStatus (bool + Persistent) – данный параметр нужен для включение \ отключение автоматического режима сборки производственной линии.
2. ManualControl (bool + Persistent) – отвечает за включение \ отключение ручного режима управления оборудованием.
3. SemiAutoStatus (bool + Persistent) – отвечает за включение \ отключение полу-автоматического режима управления оборудованием.
4. ListPosition (infotable + Persistent, DataShape создадим позже) – тут будут находиться координаты для работы в автоматическом режиме. \*Данный параметр будет зависеть от сути задания, если на сборку не влияет фактор случайности, то мы можем заполнить таблицу координатами и использовать ее для указания действий оборудованию, если же на сборку будет влиять на случайность, то нужно будет придумать алгоритм самому, чтобы заполнить таблицу координат. Универсального метода заполнения таблицы при случайном факторе сборки не существует.
5. ListGripper (infotable + Persistent, DataShape создадим позже) – такое же применение как и ListPosition, но для Gripper. Для чего это? Нам нужно отдавать команды для Gripper, только тогда, когда приедет в нужную позицию. \*Заполнение данной таблицы также зависит от фактора случайности, если он есть, то заполнение будет динамическим и метод добавления данных в таблицу нужно придумать самим, если случайности при сборки не нету, то мы просто заполняем нулями и единицами таблицу с нужным количеством шагов.
6. ListVacuum (infotable + Persistent, DataShape создадим позже) – такое же применение как и ListPosition, но для Vacuum. Для чего это? Нам нужно отдавать команды для Vacuum, только тогда, когда приедет в нужную позицию Gripper. \*Заполнение данной таблицы также зависит от фактора случайности, если он есть, то заполнение будет динамическим и метод добавления данных в таблицу нужно придумать самим, если случайности при сборки не нету, то мы просто заполняем нулями и единицами таблицу с нужным количеством шагов.
7. IndexPosition (integer + Persistent) – параметр нужен для того чтобы перемещать индекс в таблице ListPosition.
8. IndexGripper (integer + Persistent) – параметр нужен для того чтобы перемещать индекс в таблице ListGripper.
9. IndexVacuum (integer + Persistent) – параметр нужен для того чтобы перемещать индекс в таблице ListVacuum.
10. CurrentItem (integer + Persistent) – параметр будет считать какой шаг сейчас выполняет автоматический режим \*Псевдо цикл
11. Steps (integer + Persistent) – параметр предназначен для выхода из цикла.
12. DoneCode (integer + Persistent) – параметр предназначен для обозначения конца сборки изделия.

\*Если вы хотите сократить количество параметров, то можете не создавать ListVacuum, ListGripper – это можно будет реализовать через сервисы ActVacuum и ActGripper.

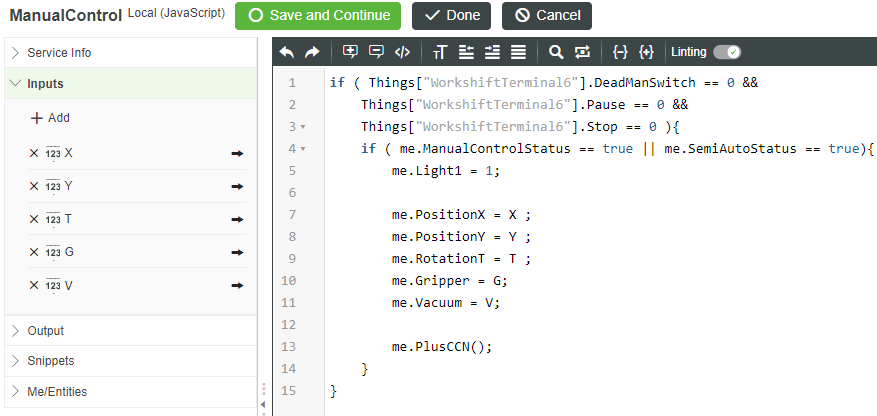
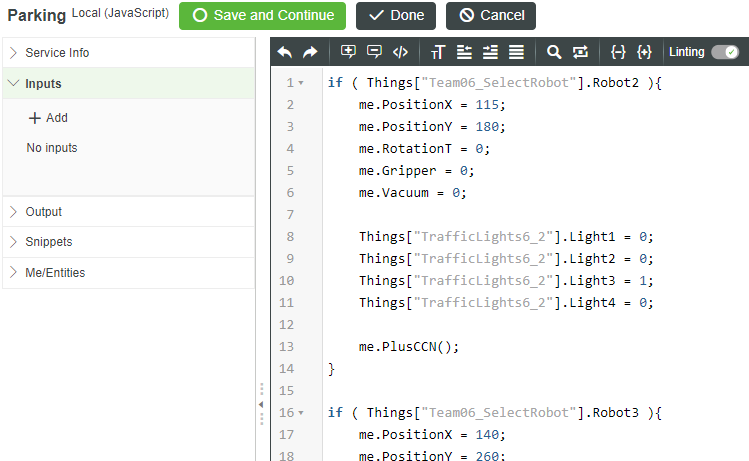
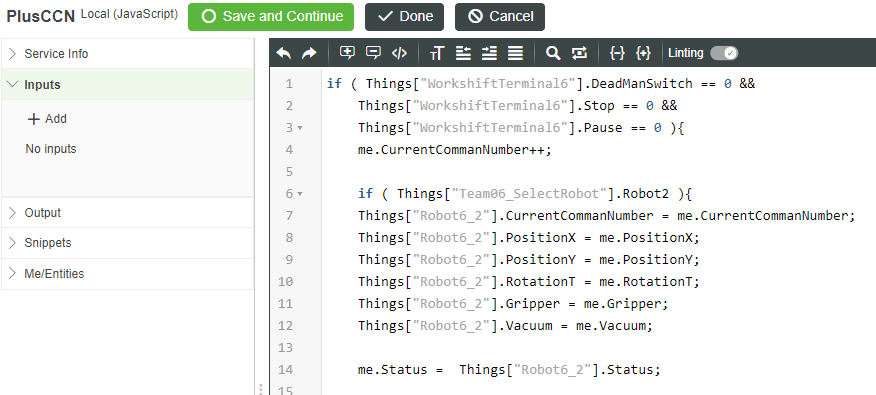
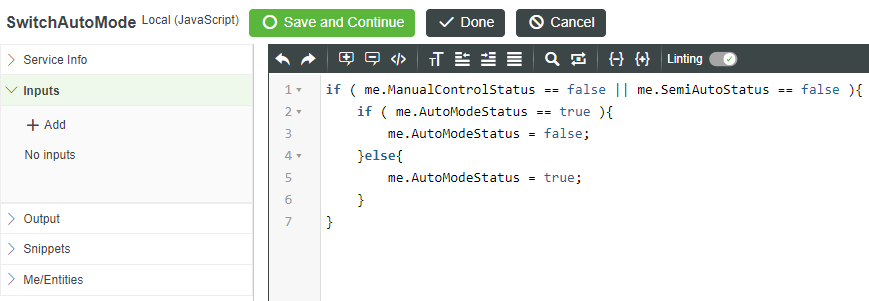
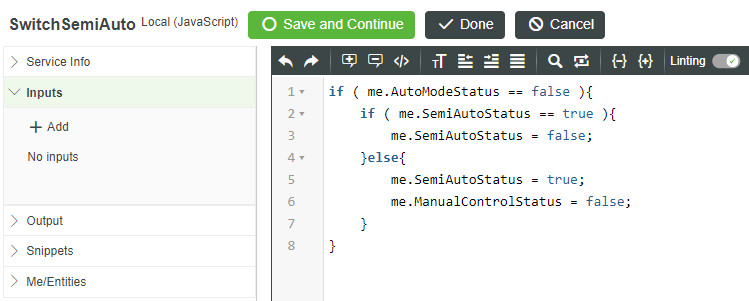
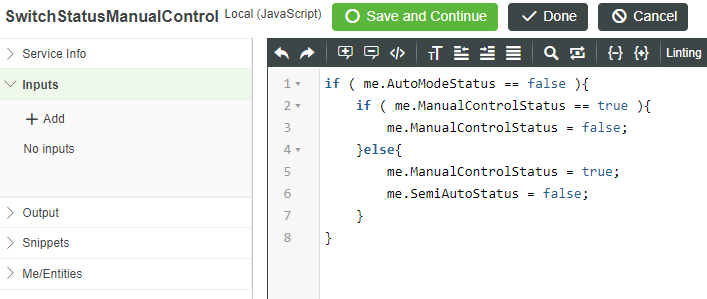


Service

Теперь разберемся с сервисами в этой вещи. Как было написано выше, теперь в данной вещи совмещены все вариации управления, а значит и сервисов стало больше.

1. ActGriper – нужен для того чтобы поднимать или отпускать Gripper. 
2. ActVacuum – нужен для того чтобы схватывать или отпускать Vacuum.
3. AutoMode – тут прописываться вся логика автоматизации.

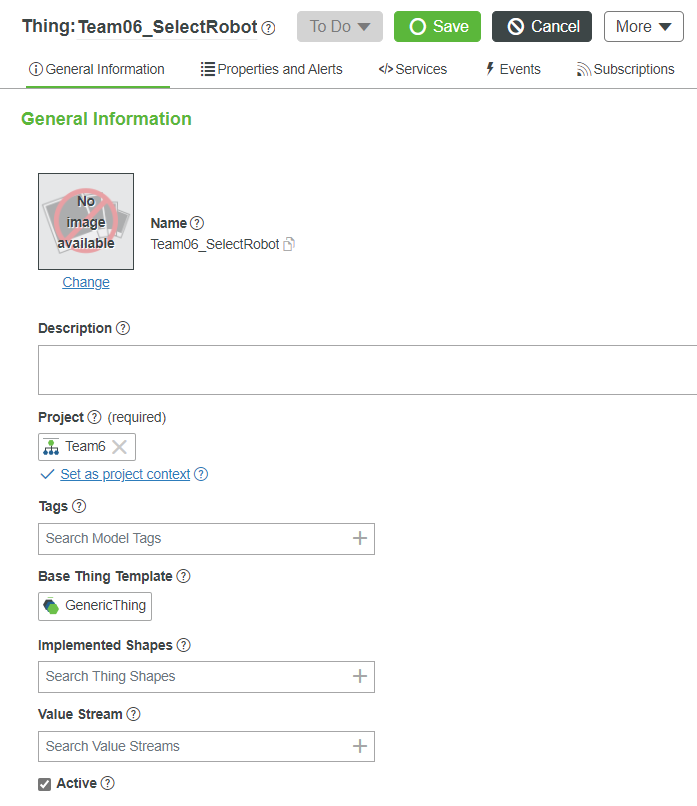
(логику опишем позже)

1. ManualControl – ручное УПРАВЛЕНИЕ, также будем использовать ее через другие сервисы, чтобы передать значения на оборудование.
2. Parking – сервис, который будет вызываться, когда робот(ы), будут заканчивать сборку, также можно будет вызвать данный сервис в ручную через Mashup. (Код повторяется для каждого робота, сначала вы проверяете какой робот выбран, а после задаете ему его значение парковки)
3. PlusCCN – увеличение CurrentCommandNumber, ибо если данный параметр не увеличить, то робот не будет двигаться. Данный метод не внедрен в ManualControl и другие сервисы для управления, потому что сначала вводимые параметры должны проверяться на критические значения, а после передаваться на оборудование, если значения параметров в норме. 
4. SwitchAutoMode - переключение автоматического режима. 
5. SwitchSemiAuto – переключение полуавтоматического режима. Сами сервисы для перемещения робота, будут в вещи SemiAuto, чтобы не нагружать количеством сервисов SmartControl. 
6. SwitchManualControl - переключение ручного режима. 

### Глава 13. Выбор робота для управления

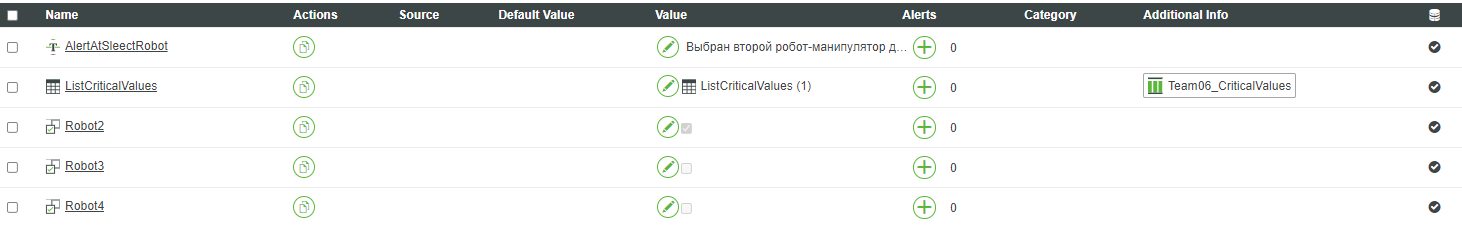
В вашем проекте может быть несколько робот для управления с помощью ручного управления, полуавтоматики и полностью автоматической сборки. Мы помним, что роботом управляем через SmartControl, а значит, когда мы выбрали робота, мы должны передавать значения выбранного робота в данную вещь, чтобы на Mashup знать, где находится робот и вводить верные координаты при ручном управлении.

Для того чтобы мы могли выбирать робота, нам нужно создать вещь «SelectRobto», через которую мы будем указывать каким роботом мы хотим управлять.

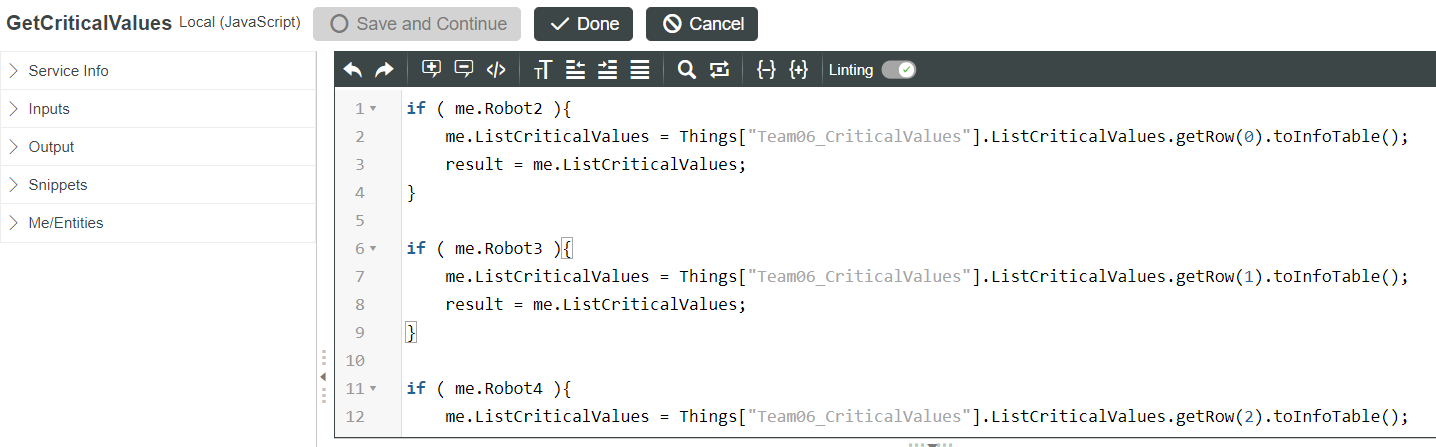


Properties

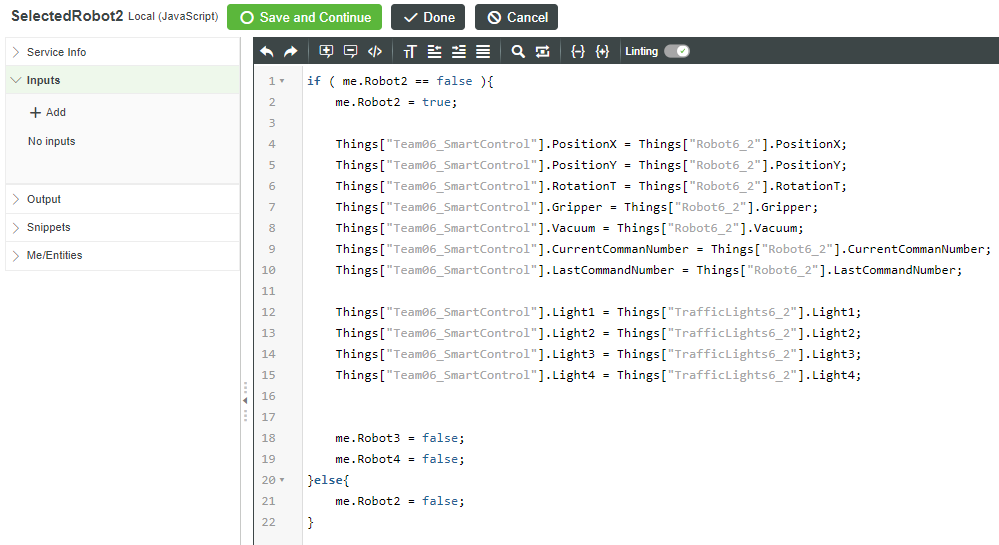
1. AlertAtSelectRobot(string + Persistent) – нужен для визуального отображения текста выбранного робота.
2. ListCtiticalValues (infotable + Persisnetr, пока создаем без DataShape, вернемся позже) – нужен для того чтобы создать проверку вводимых значений и критических переменных выбранного робота.
3. Robot [количество роботов на производственной линии] (bool + Persistent) – данный параметр предназначен для выбора робота.



Services

«GetCriticalValues» (Получить критические значение выбранного робота, автоматически) 

«SelectRobot» (Делается под каждого робота свой сервис для выбора)



Subscription

«AlertLabel» (Автоматически будет писать на Mashup, какой выбран робот)



### Глава 14. Полуавтоматическое управление

Полуавтоматическое управление заключается в том, что оператор может нажимать одну кнопку на Mashup и оборудование будет это выполнять. Ранее мы сделали включение \ отключение данного режима в вещи SmartControl, так же там добавили сервисы «ActGripper» и «ActVacuum», данные два сервиса нужны для выполнения двух действий. Gripper – подняться и отпустить, что он будет делать будет исходить от его реального положения, точно такой же принцип работы у Vacuum.

Теперь нашей задачей стоит создание кнопок перемещения по координатной плоскости. Что это такое? Возьмем в пример данную схему:

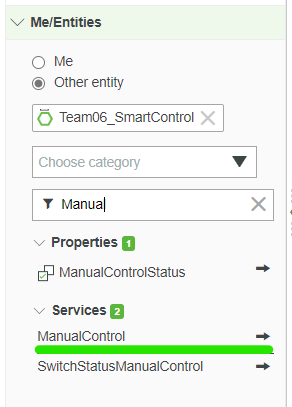
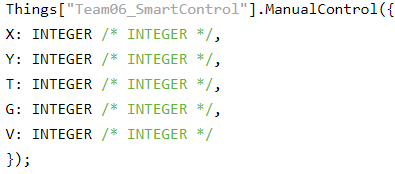
Как видите есть сектора обозначенными цифрами и буквами, внутри сектора находятся кружки и наша цель, создать на визуальном интерфейсе интуитивно понятные кнопки, на которые пользователь будет нажимать, и робот будет передвигаться в нажатую им позицию.

Как это сделать? Все довольно очевидно, нам нужно сделать столько же сервисов, сколько количество ключевых позиций у координатной плоскости. Чтобы нам не загромождать кучей сервисов SmartControl, мы будем все пихать в отдельную вещь под именем SemiAuto, также это нам даст большое преимущество в сборке визуального интерфейса, потому что мы не будем путаться в сервисах и все будет находиться под рукой в одном.

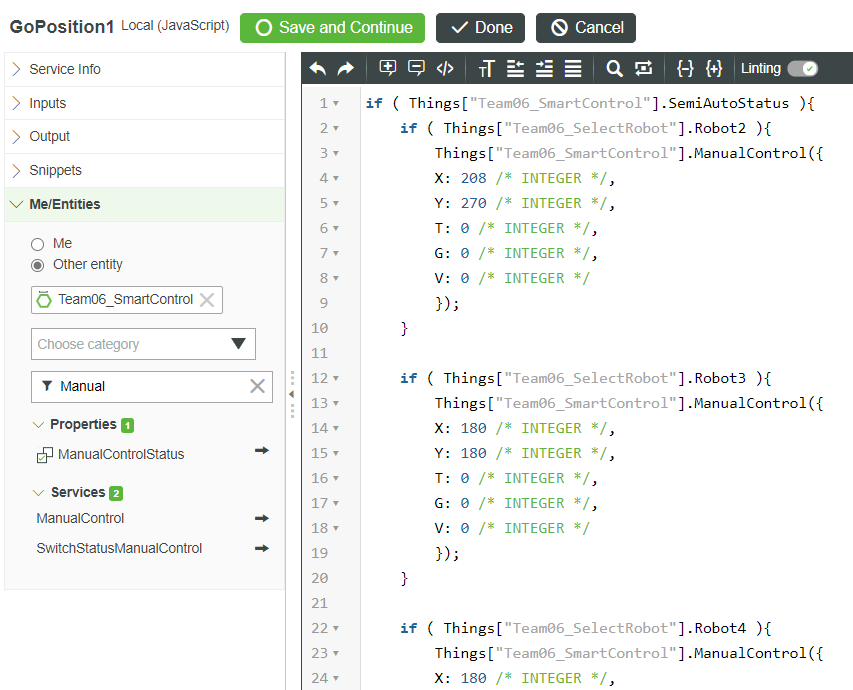
И так приступим к созданию сервисов. Помните ранее мы в SmartControl создавали такой сервис как «ManualControl»? Так вот, сейчас он нам очень поможет и нам не придется каждый раз прописывать логиику передвижение робота, а только использовать ранее созданный сервис и передавать нужные для нас значения XYT.

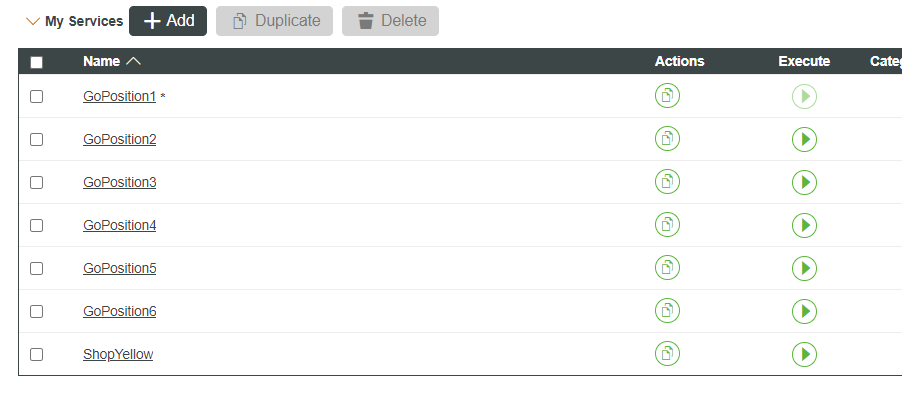
Создаем сервис с именем «GoPosition1», что исходя из имени обозначает передвижение робота в целевую позицию 1 на координатной плоскости.

Для начала мы проверяем включен ли у нас статус SemiAuto в вещи SmartContol, также узнаем какой робот был выбран, потому что для каждого робота будут разные значения координат, а дальше просто вызываем сервис «ManualControl» в который помещаем нужные нам координаты. \*Проверка RemoteTerminal находится в ManualControl

\*Чтобы легко вызывать сервис и не писать его руками, вы можете воспользоваться вкладкой Me/Entities  
Дальше мы получим такой снипет, в котором вместо INTEGER, нужно вписать нужные нам координаты:  


Так выглядит полностью написанный сервис:



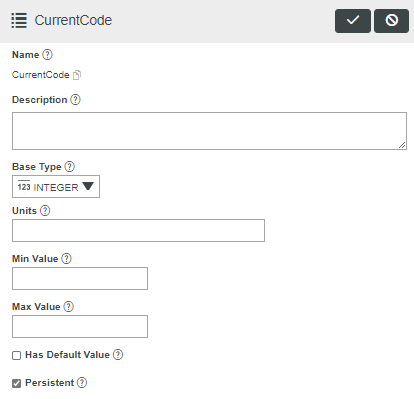
И как писалось ранее, создаем ровно столько же сервисов – сколько количество позиций на координатной плоскости.

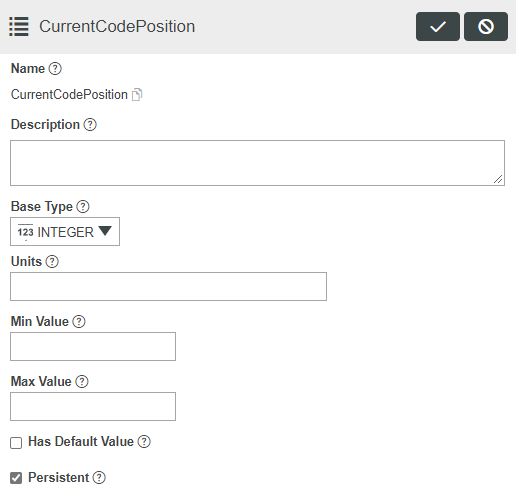
### Глава 15. Полная автоматизация производтсвенной ячейки

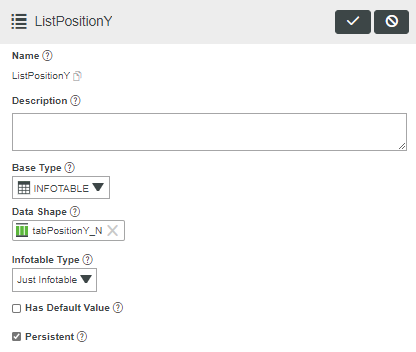
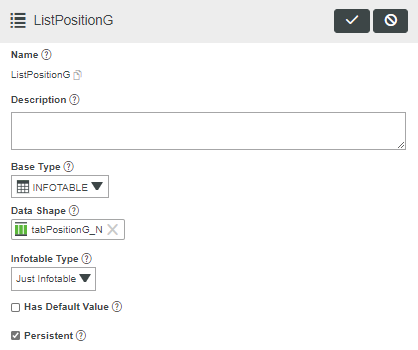
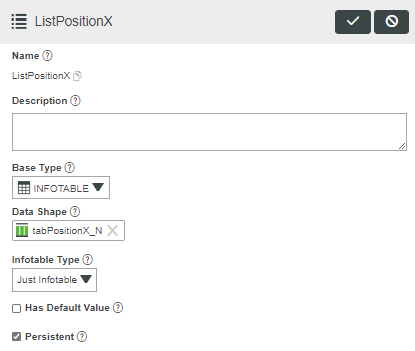
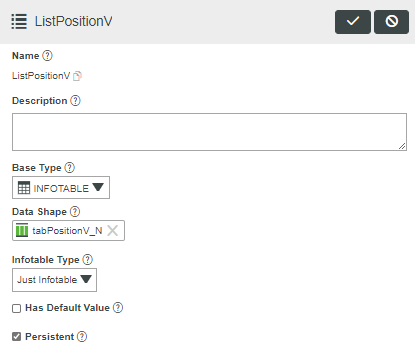
Это будет самая объемная глава по ThingWorx, прочитав эту главу вы сможете сделать полностью автоматизированное управление производственной ячейкой. Чтобы реализовать автоматизацию, нужно понять, как робот исполняет команды, посланные ему.

Для начала нужно подготовить нашу вещь «Auto», создать в ней кучу параметров, переменных и таблиц, с которыми мы будем работать в последующем.

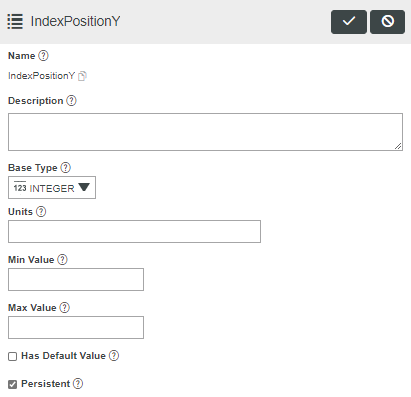
Сначала создадим таблицу куда нам будут приходить коды сборок со смарт камеры. Имя таблицы будет «CodeList», базовый тип выбираем «INFOTABLE», Data Shapes выбираем ранее созданный с именем «CodeShape» и ставим галочку в пункте «Persistent».  

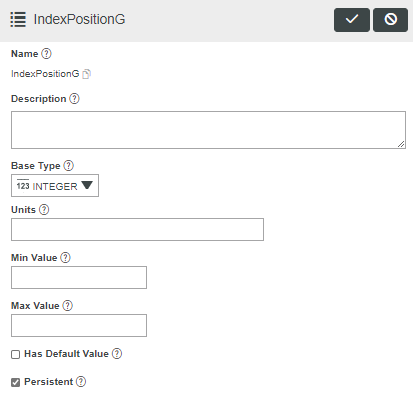
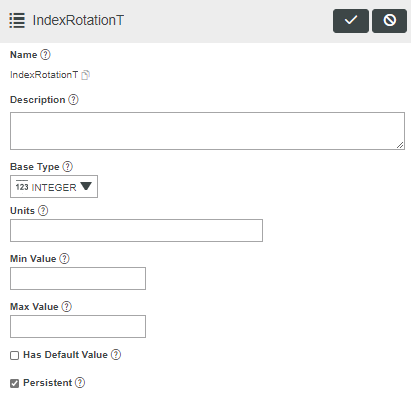
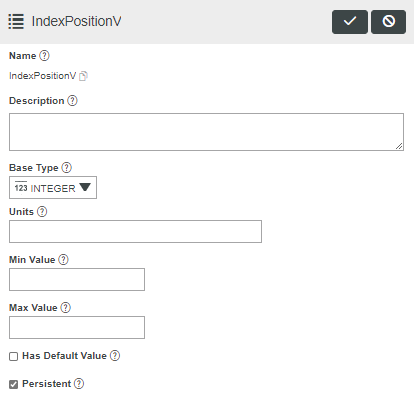

Дальше создаем параметр, который будет брать значение из таблицы и даем ему имя «CurrentCode», базовый тип «INTEGER» и галочка в пункте «Persistent».  


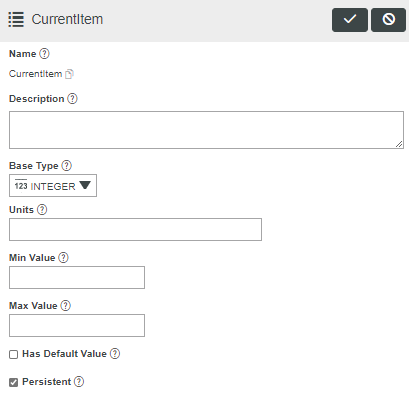
Создаем параметр, который будет показывать индекс выбранного кода из таблицы, даем ему имя «CurrentCodePosition», базовый тип «INTEGER» и галочка в пункте «Persistent».  


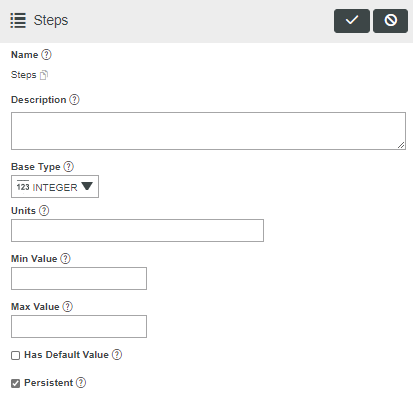
Теперь нужно создать 5 таблиц, в которых будут храниться координаты перемещения роботов.   
 

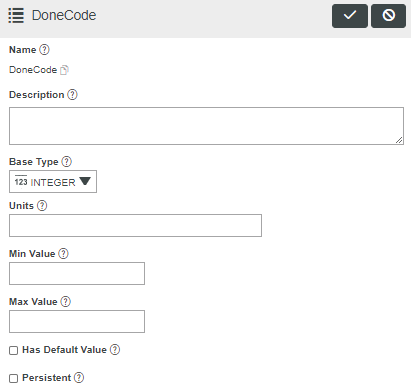
Теперь чтобы перебирать значение из таблицы нужно сделать переменные, которые будут выставлять следующие значение после каждого выполненного действия.

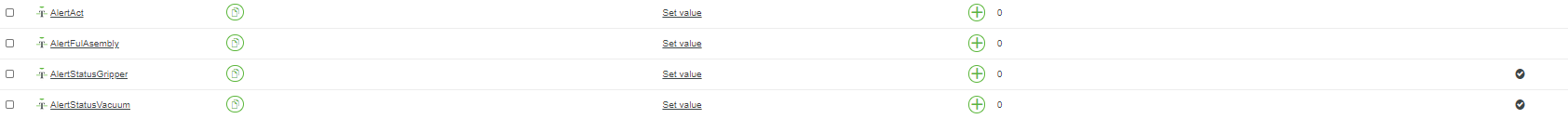


  
  
Осталось сделать немного новых параметров, которые помогут нам в реализации автоматизации.

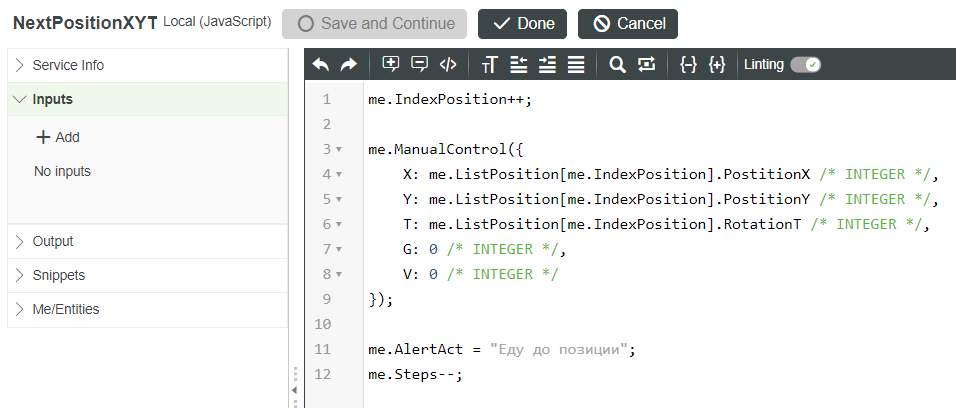
С помощью «CurrentItem» мы будем создавать псевдо цикл, который будет вызывать сервисы для передачи координат роботу из таблиц до тех пор пока не закончатся шаги сборки.  


В параметре «Steps» будут отсчитываться количество шагов робота до конца сборки.  


Так же нужно создать параметр, который будет показывать о законченности сборки изделия.  


Дополнительные параметры для отображение текущего состояния процесса автоматизированного процесса.  


Давайте напишем сервис, который будет вызываться после каждого нового шага робота и устанавливать новые координаты в параметры управления взяв их таблицы с координатами, имя сервиса будет «NextPositionXYT». Мы будем использовать метод для ручного управления и передавать туда нужные нам координаты, это сделано для упрощения и избежание повторного написания кода управления.



Сервисы для управления Gripper и Vacuum уже написаны, имеют имена «ActGripper» и «ActVacuum». Мы просто их будем вызывать в сервисе «AutoMode».

Автоматическую работу ламп реализуем с помощью Status и подписок в вещи «SmartControl»

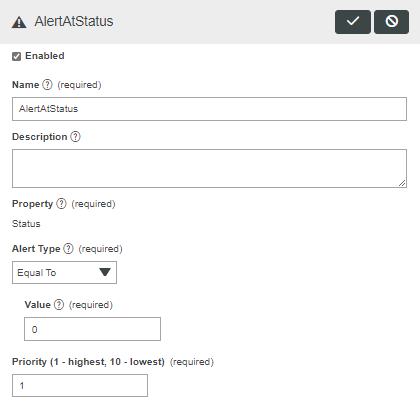
У робота есть параметр «Status», что это такое? Данный параметр имеет значение 1 и 0, когда робот исполняет команду, то свойство параметра будет отображать 1, когда робот закончит выполнение команды, свойство параметра будет равно 0. Это нужно для того, чтобы мы понимали, когда робот закончил исполнять посланную ему команду (переместится, опустится, схватить), но как нам заставить послать новую команду для робота, когда он закончил выполнять действие?

Для того чтобы программа понимала, когда нужно снова запустить сервис выполнение команды у нас это будет «SmartControl», да в вещи «SmartControl» у нас будет сервис «SmartControl», авторы данного пособие более логичного названия не смогли придумать.

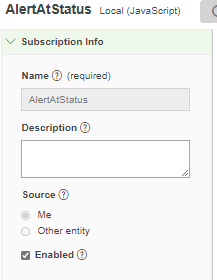
Создаем сервис в «SmartControl» в котором прописываем всю логику работы автоматизированного процесса. Давайте для начало опишем наш сервис автоматизации. В самом начале проверяется включено ли производство, не активирована аварийная ситуация, не стоит ли пауза или стоп, также не е активировано полуавтоматика или ручное управление. После срабатывает сервисы TrafficLightStartAsembly для того, чтобы показать, что началась автоматическая сборка детали, дальше прибавляем к CurrentItem + 1 чтобы активировать наш псевдоцикл и ставим параметру DoneCode = 0 – это обозначает, что сборка изделия не завершена. Дальше прописываем наш псевдо цикл, если CurrentItem == 1, то выполняем сервис передвижения, дальше когда сервис вызовется, то он прибавит к CurrentItem + 1 и будет выполняться второе условие при котором CurrentItem == 2 – это отпустить или поднять гриппер в зависимости от того какие координаты он получит из таблицы, вновь вызывается сервис меняется CurrentItem и срабатывает другое условие при котором он CurrentItem == 3 – это захватить или отпустить схватам деталь, при CurrentItem == 4 будет гриппер подниматься либо отпускаться в зависимости от приходящих координат и наконец-то пятый CurrentItem – при котором будет передвижение робота на новые координаты и перебрасывать его вновь на первый шаг, все это время у нас в сервисах отсчитываются шаги чтобы рано или поздно выйти из этого цикла, как задать шаги и начальные координаты мы разберем позже – это будет создание дополнительных циклов, которые будут обрабатывать входящий код из смарт камеры и задавать нужные свойства для выполнения указанного кода сборки действия. После как шаги отсчитают до 0, то CurrentItem тоже станет равен нулю и больше не будет вызываться наш цикл, выставляем DoneCode = 1, так как сборка завершена успешно, вызываем сервис о том который покажет на светосигнальной лампе, что процесс сборки завершен и робот стоит на парковке при этом вызывается сервис, который отправляет робота на парковку и через Alert параметры мы покажем на визуальном интерфейсе, что задача выполнена.



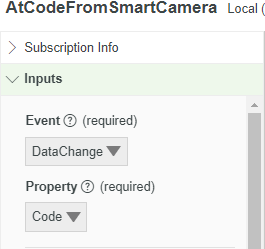
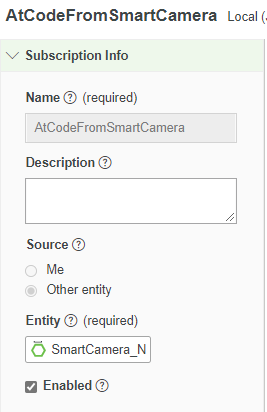
Теперь в каждом созданном роботе добавляем Alert во вкладке «Properties and Alerts»  
   
Но перед этим, что такое «Alert»?   
Оповещения (Alert) — это особый тип событий. Оповещения позволяют определить правила для запуска событий. Как и события, вы должны определить подписку для обработки изменения состояния. Все свойства в форме вещи, шаблоне вещи или вещи могут иметь одно или несколько определенных условий оповещения. При выполнении условия оповещения ThingWorx создает событие. Вы можете подписаться на событие и определить ответ на предупреждение с помощью JavaScript.

Тут выбираем «Alerts». Далее добавляем «Alert» с данными параметрами:  
Даем имя – «AlertAtStatus», в пункте «Property» выбираем статус «Status» робота, в «AlertAtStatus» выбираем «Equal To», значение выставляем «0», приоритет «1» (по умолчанию). Как должен выглядеть ваш «Alert».  


Теперь нужно подписаться на наше оповещение, чтобы, когда робот выполнил действия и его статус изменился на 0, то сработает оповещение для подписки и подписка вызовет сервис «Auto» для продолжения последовательность действий робота.

Создаем подписку в самом роботе и называем ее так же, как оповещение.  
 

Теперь нужно реализовать через подписку добавление нового кода в таблицу при изменении свойства параметра «Code» в вещи «SmartCamera»

Создаем подписку и называем ее «AtCodeFromSmartCamera», в пункте «Source» выбираем «Other entity» и выбираем вещь смарт камеры, не забываем включить подписку. Далее в «Inputs» в пункте «Event» выставляем значение «DataChange», после выбираем параметр «Code».  
Теперь опишем логику подписки. При изменении свойства в смарт камере, будет вызываться данная подписка, которая проверяет не включена ли полуавтоматика или ручное управление, если нет, то создается объект с параметрами, как у нашей таблице «CodeList» и собирает строки для добавления информации в нее. Первый столбец – это полученный код со смарт камеры, второй столбец – это время получения данного кода, далее он собирается и добавляется в таблицу. После происходит проверка включена ли полуавтоматика или ручное управление, далее проверяется выбран ли какой-то робот, далее идет проверка на то выполняет ли робот действие, закончен псевдо цикл, не стоит стоп и включена автоматическая сборка, если условие верно, то вызывается сервис GetNewCode(), который мы создадим ниже, но добавим сейчас. Полная реализация кода:



Теперь создаем сервис GetNewCode, который будет присваивать параметру CurrentCode последние значение приходящие в смарт камеру. Создавая сервис, ничего дополнительного в него добавлять не нужно, речь идет о вкладках «Inputs» и «Output». Описание сервиса – создаем переменную, которая получит число кол-во записей в таблице, далее происходит проверка, если CurrentCodePosition меньше кол-во записей в таблице, то мы прибавляем +1 к CurrentCodePosition и приравниваем параметр CurrentCode последнему значению таблицы и вызываем сервис ChecCode, который мы создадим ниже, но добавим сейчас.

Полная реализация данного сервиса:

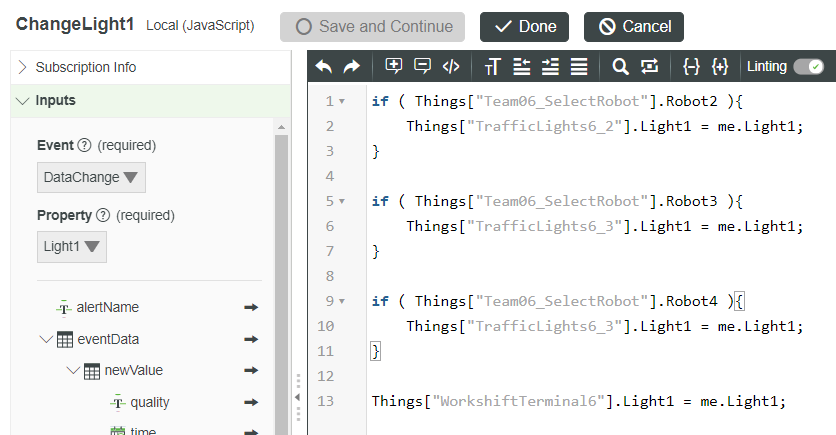


Далее создаем сервис CheckCode, который будет проверять потупленный код со смарт камеры и уже задавать нужные индексы в ранее заготовленные таблицы с координатами, а также будут устанавливать для Alert доп. информацию, которая будет выводится на визуальный интерфейс.   
Описание сервиса – производится проверка на существование кода, далее задается индекс с каких координат должен стартовать сборку робот, выставляется CurrentItem на ноль, для того чтобы, если при автоматической сборки была активирована кнопка стоп и отправлен новый код сборки, поле задается в переменную «Steps» количество шагов для выполнения кода сборки (вы должны заранее просчитать сколько шагов сделает робот, чтобы закончить данную сборку), в Alert отправляется схема последовательности сборки и в самом конце вызывается сервис AutoMode(), чтобы началась автоматическая сборка детали.

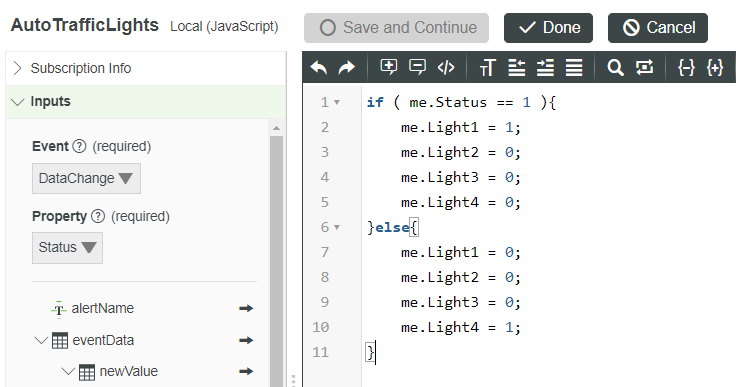
Полная реализация данного сервиса (количество проверок зависит от вашего задания, сколько дано вам кодов для сборки):



Чтобы сделать автоматическое переключение TrafficLights, нужно создать подписки переключений для каждой, а после создадим подписку, которая будет автоматически управлять каждой из них ориентируясь на Status робота.



Теперь создаем подписку, которая будет переключать значения в SmartControl, а подписки, которые сделали выше для каждой лампы будут переключать в лампах работающего робота и удаленного терминала.



### Глава 16. Доработка удаленного терминала

После создания автоматизации, нужно реализовать управление ею через удаленный терминал.

Создаем сервис «DMSIndicator», который будет переключать светосигнальную лампу при включение аварийной ситуации.



Также нужно создать индикатор лампы при включении паузы или стоп – «PauseIndicator».



Запуск автоматической сборки «StatusAsembly».

if ( Things["RPManagement\_N"].Status == 1 ){

    if ( me.Button1 == 0 ){

        me.Button1 = 1;

    }else{

        me.Button1 = 0;

        Things["SmartControl"].CurrentItem = 0;

        Things["SmartControl"].Steps = 0;

        Things["SmartControl"].AlertAct = "Сборка выкл";

        Things["SmartControl"].AlertFulAsembly = "Сборка отключена";

    }

}

Создаем подписку «PuseStatus» и привязываем «DataChange» на параметр «Button2». Подписка будет ставить произовдство на паузу и транслировать данный статус на светосигнальную лампу.



Создаем подписку «StopStatus» и привязываем «DataChange» на параметр «Button3». Подписка обнуляет ходы автоматизации и будет ждать новый код сборки, после отжатия стопа, производство начнется с новой сборки.



Создаем подписку «» и привязываем «DataChange» на параметр «DeadManSwitch». После того как кнопка DMS будет отключен, производство продолжится.



### Глава 17. Организация Сбора Данных

Одной из важнейших задач является грамотный сбор, обработка и визуализация данных от оборудования производственной ячейки.

Необходимо выполнить несколько задач, связанных с данными:

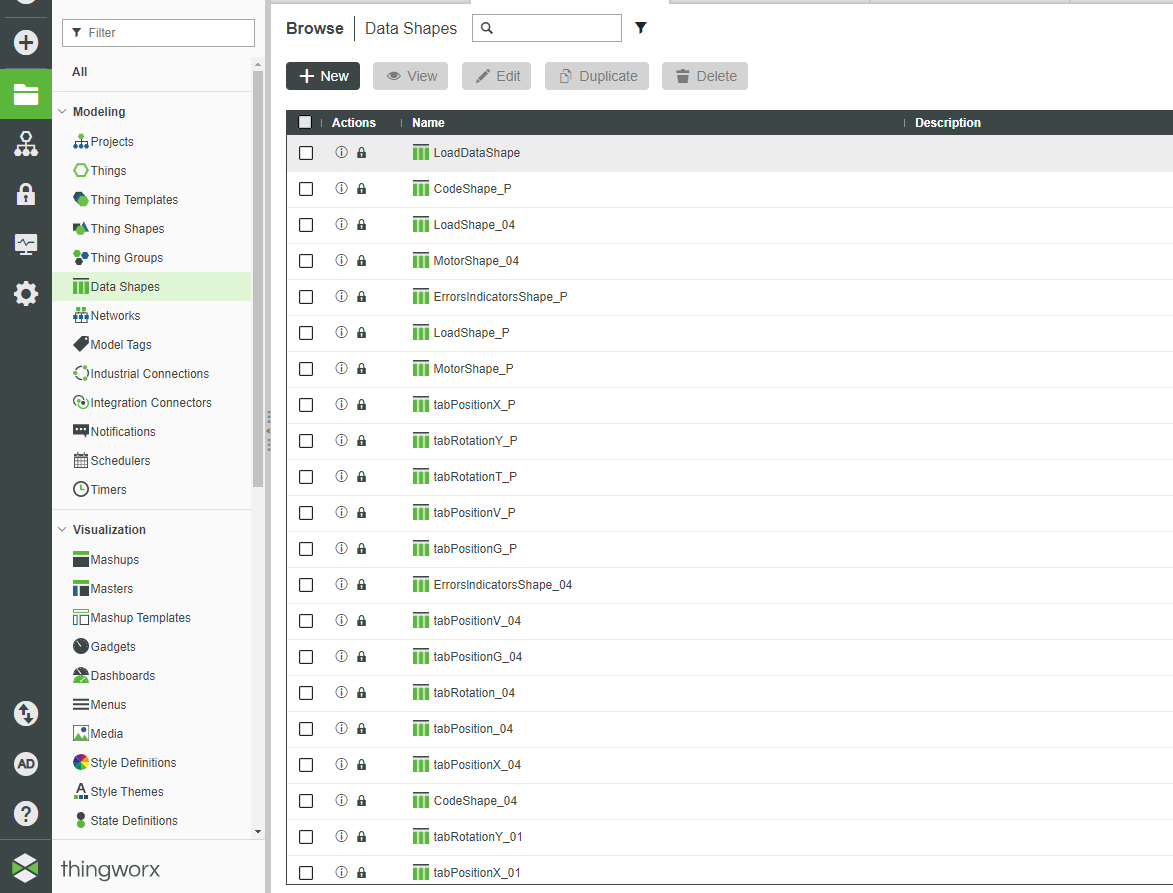
* Разработка веб-интерфейса автоматизированного рабочего места инженера технолога (создание интерфейсов будет рассмотрено в следующих главах);
* Организация вывода данных, полученных от оборудования. Существенным является и период времени от изменения состояния робота до отображения изменений на интерфейсе;
* Реализация возможности ввода в интерфейсе критических и допустимых значений параметров оборудования;
* Реализация сохранения и отображения исключительных ситуаций (неправильный код изделия, выход значений на допустимые диапазоны, перегрев сервомоторов роботов, превышение нагрузки, получение недопустимых команд);
* Организация настраиваемых сохранений данных мониторинга функционирования оборудования гибкой производственной ячейки (настройка периода или частоты сохранения, списка конкретных параметров, исключение сохранения неизменных величин, включения-отключения сохранения);
* Разработка средства улучшения восприятия информации, поступающей от оборудования;
* Реализация двустороннего обмена данными с удаленным терминалом и веб-интерфейс, отображающий состояние элементов управления терминала и позволяющий управлять сигнализацией на терминале.

### Глава 18. Создание DATA SHAPE

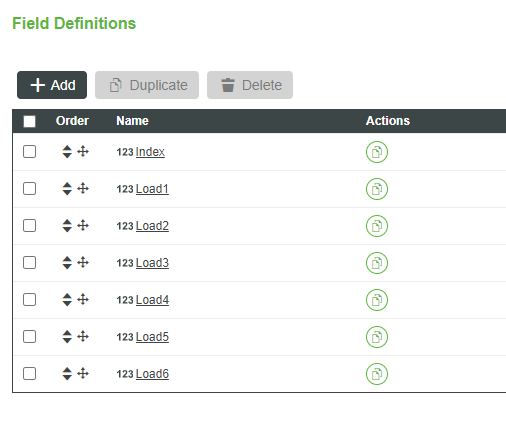
Процесс создания логгирования – достаточно долгий и однообразный процесс. Копирование тут не пройдет, могут потеряться связи.

В данном документе в качестве примера будет продемонстрирован процесс создания записи данных для нагрузки роботов (Load).

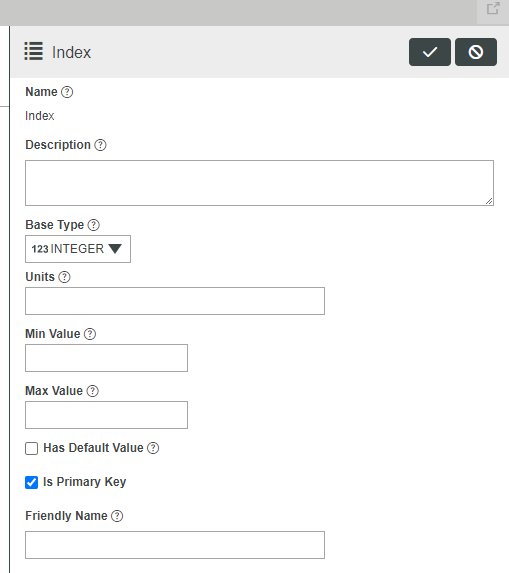
Все остальное делается по такому-же принципу, поэтому нет необходимости описывать всё.

Для начала, заходим во вкладку DATA SHAPES, и создаем его.

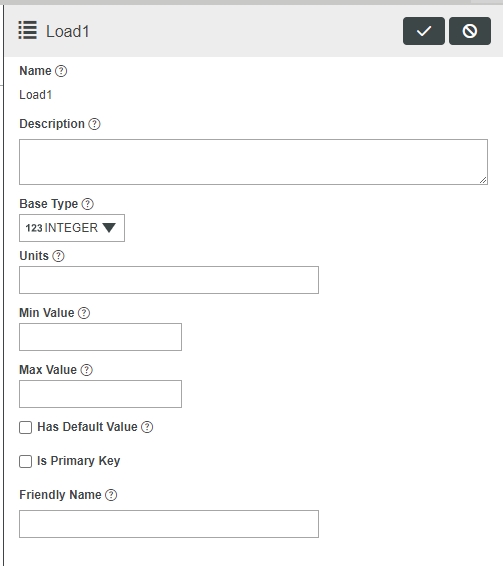
Все «DataShape» можно называть как вы хотите, но мы рекомендуем держаться строгого нейминга по типу: “LoadDataShape”.

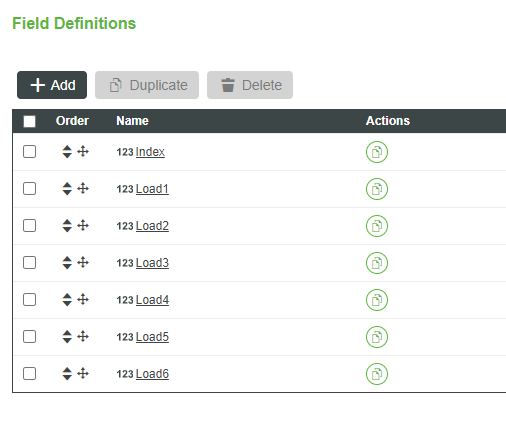
Заходим во вкладку «Field Definitions» и заполняем поля следующими свойствами:

«Index» является ключевым полем, поэтому нужно поставить галочку на свойстве «**IS PRIMARY KEY»:**



Далее, это галку ставить уже не нужно:



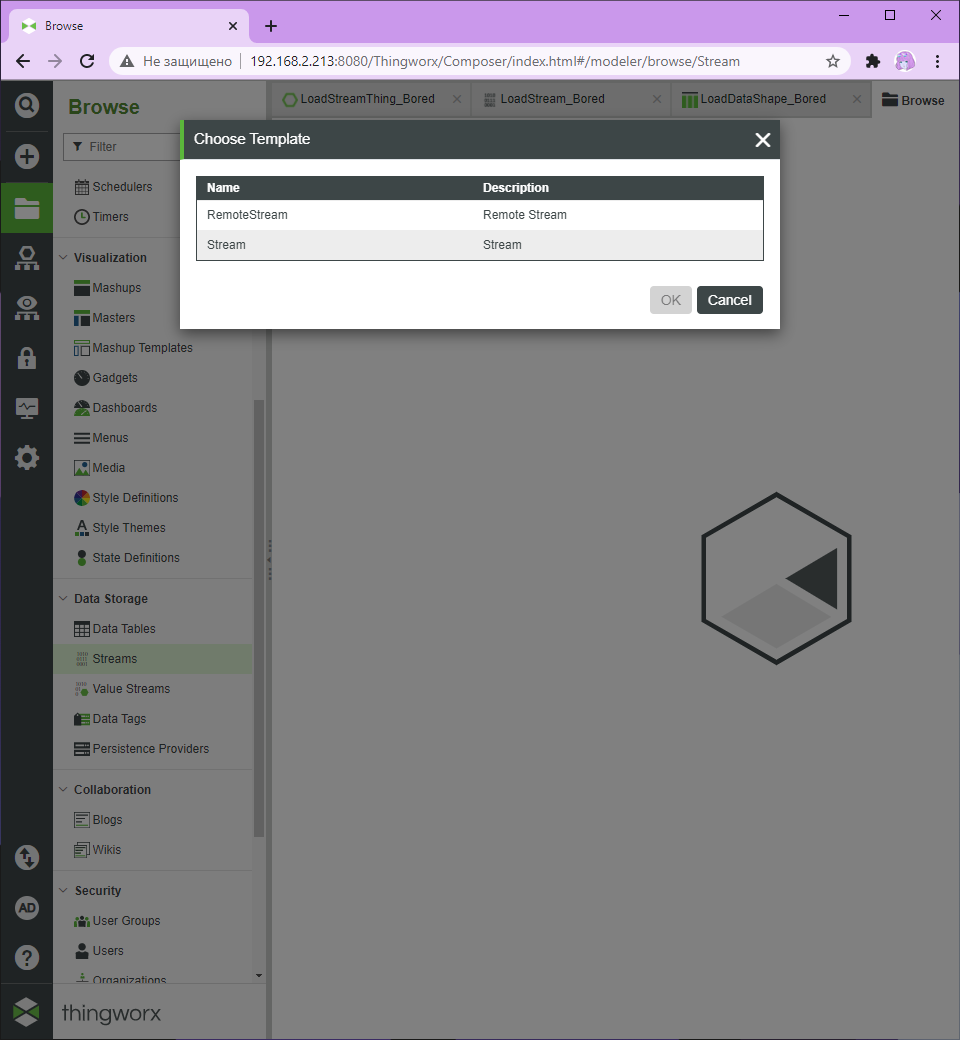


Сохраняем и закрываем.

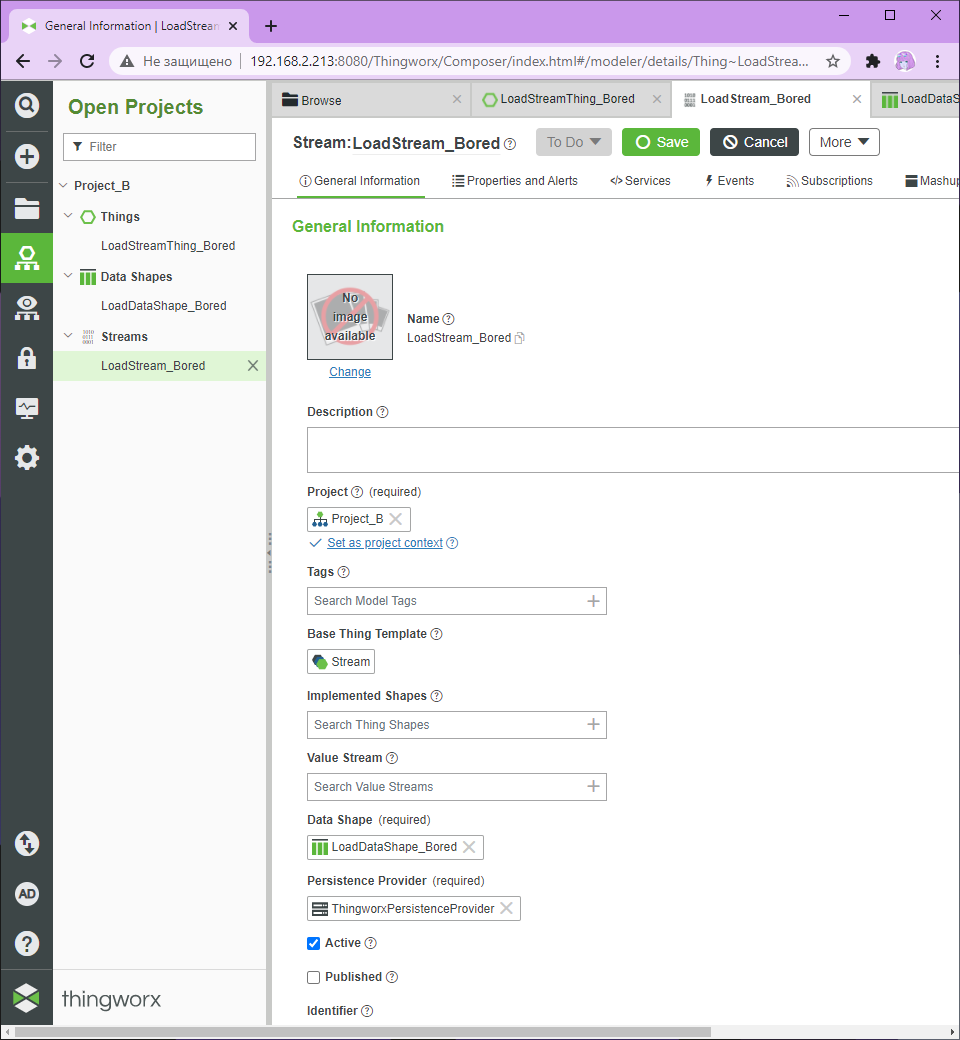
Не нужно создавать несколько DataShape для нескольких роботов. То-есть достаточно создать Load;Temp;Control для всех роботов по заданию.

### Глава 19. Создание Stream

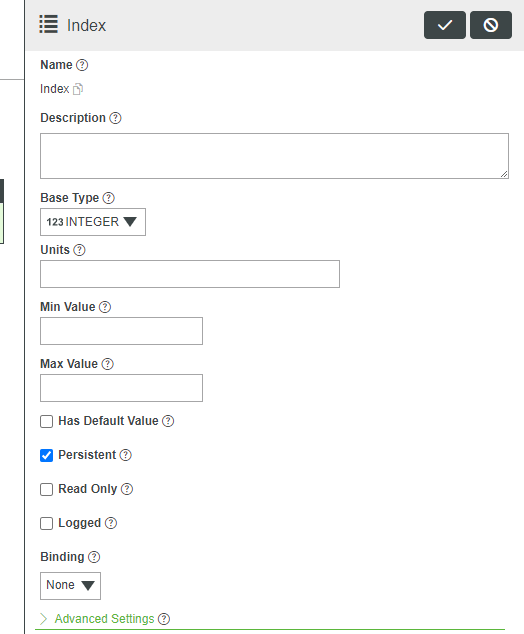
Переходим в раздел «Data Storage», а именно, «Streams».



Создаем новый Stream. Далее, выбираем проект, а также DataShape, который был создан выше.

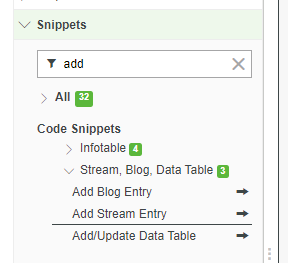


Переходим во вкладку «Properties». Тут задаем одно свойство – «Index».



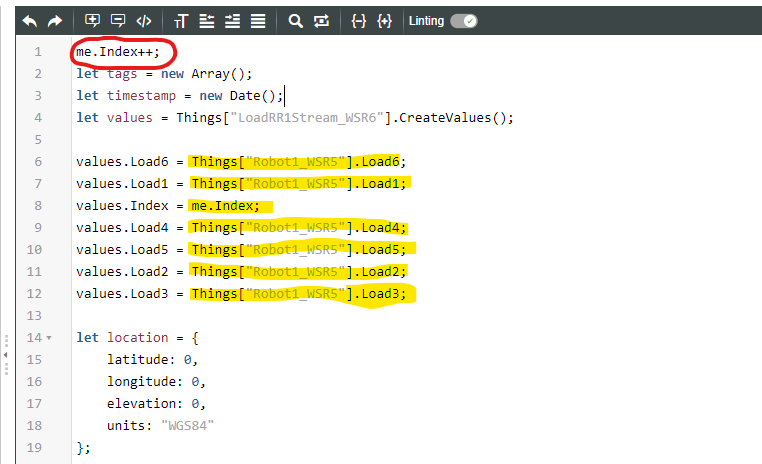
Дальше, создаем сервис, который мы назвали addstr(Короткое сокращение от AddStreamEntryService).

Выбираем сниппет:



И в окне выбора, выбираем созданный нами стрим.

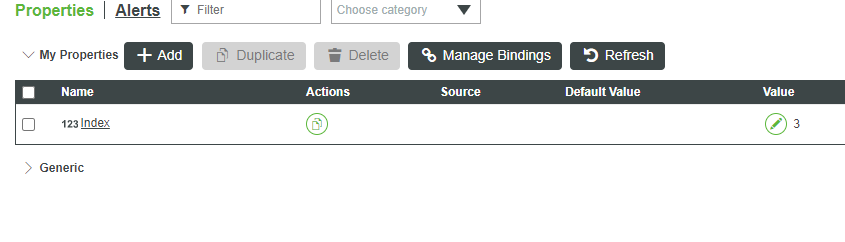
Далее, редактируем сервис. В самом начале реализуем прибавление индекса, а снизу, мы заполняем стрим значениями из необходимого робота. Индекс приравниваем к индексу, созданному ранее. Вот, что должно получится в итоге:

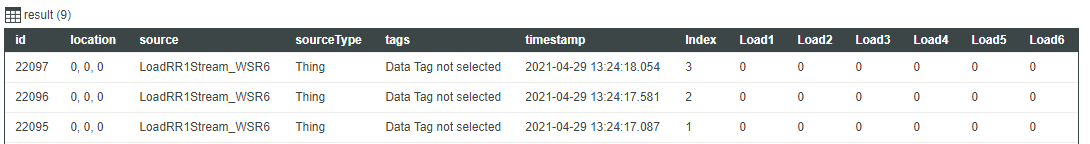


Чтобы проверить работоспособность сервиса, активируем его (желательно несколько раз).



Затем, переходим во вкладку «Properties». Я активировал сервис три раза, значит и запись должна была сработать три раза:



****Чтобы проверить запись данных, снова переходим в сервисы, и находим готовый сервис «QueryStreamEntriesWithData». Активируем, и видим что нам вывелось 3 готовых лога:

Работает это так: как только прибавляется индекс, наш стрим берет данные из робота и записывает это в себя. Чтобы реализовать логгирование, нужно просто в вашем коде расположить наш сервис в необходимых местах.

В итоге, вам нужно создать столько стримов и даташейпов, сколько необходимо.

Для примера, допустим по заданию у нас есть два робота. Какие данные нам необходимо записать?

*Данные о нагрузке сервомоторов:*

*R1 и R2 означают соответственно первого и второго робота.*

* *LoadR1Stream*
* *LoadR2Stream*

*Данные о положении сервомоторов:*

* *MotorR1Stream*
* *MotorR2Stream*

*Данные о температуре сервомоторов:*

* *TempR1Stream*
* *TempR2Stream*

*Данные о управлении роботом:*

* *ControlR1Stream*
* *ControlR2Stream*

### Глава 20. Контроль качества изделий и его логгирование

Во время работы на производственной ячейке, роботы выполняют некоторое количество сборок. Контроль качества изделий – специально созданная вещь, в которой будет все это хранится.

По заданию, робот может совершить правильную сборку, брак, или пришел неверный код сборки. Контроль этот по заданию, учитывается автоматически и в ручном режиме. Ручной режим – специальный мэшап с тремя кнопками (Right, Defect, Wrong). Пользователь сам решает, когда сборка выполнена правильно, а когда нет. По заданию, каждая из возможных выполненных сборок (верно, брак, неверный код) должна логгироваться (ручная и автоматическая).

Для всего этого, потребуется создать три вещи:

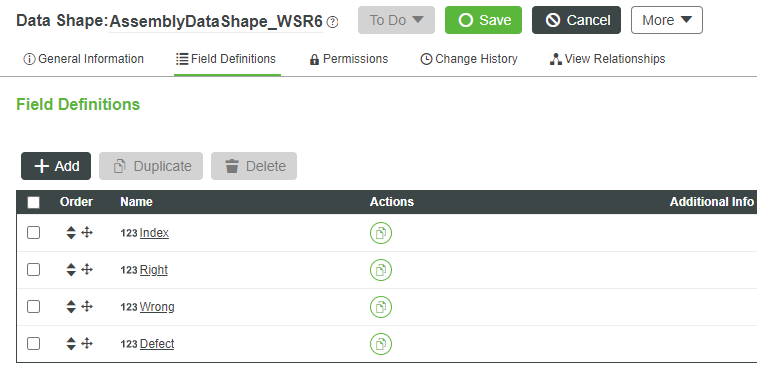
«HandAssembly» – хранение ручного контроля;

«AutoAssembly» – запись данных автоматически;

«JointAssembly» – объединение записей ручной и автоматической сборок.

### Глава 21. Логгирование сборок

Первый DataShape будет для ручного и автоматического контроля качества изделий.



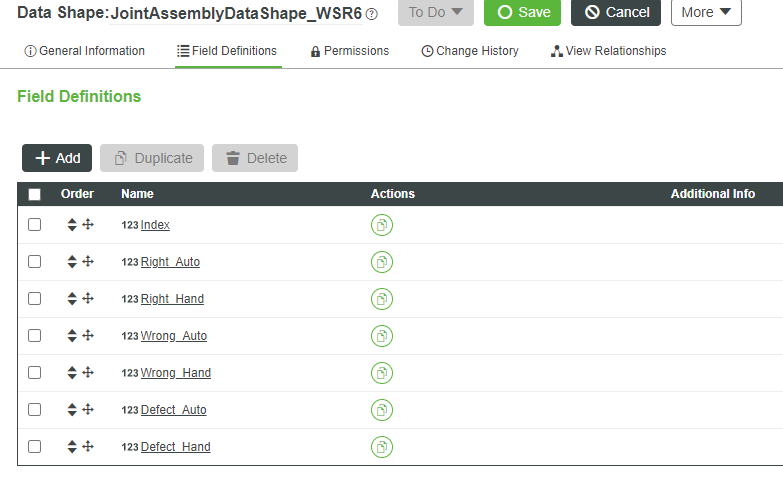
Index все также имеет параметр «Primary Key».

Right – правильная сборка;

Wrong – неверный код изделия;

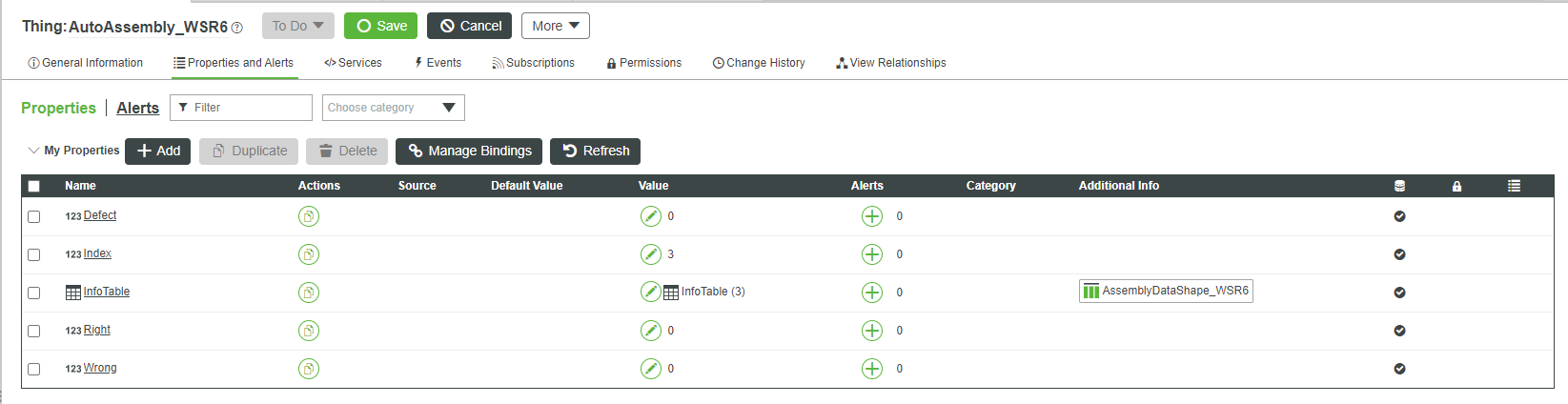
Defect – брак.

Второй DataShape – объединенный учет и автоматической и ручной сборки. Тут будут хранится все данные.

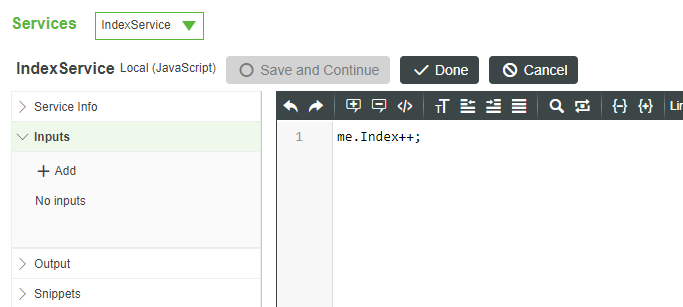


Далее, создаем уже вещь – AutoAssembly, которая будет автоматически записывать данные.

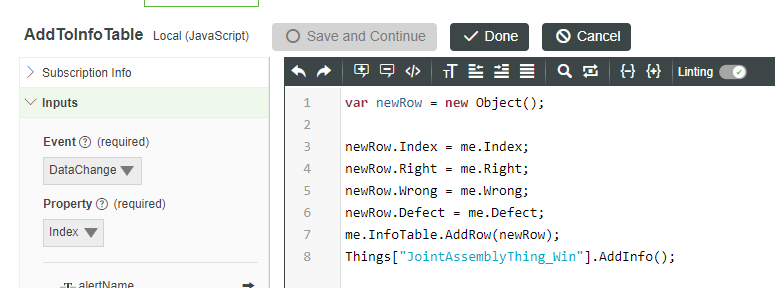
Тут у нас есть параметры Index,Defect,Right,Wrong, а также InfoTable – таблица типа данных “infotable”, которую мы связываем с AssemblyDataShape, который мы создали выше.



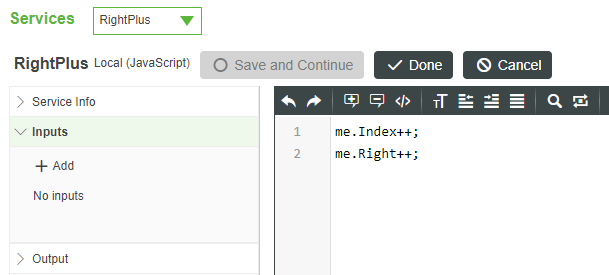
Далее, создаем сервис, где реализуем прибавление индекса.

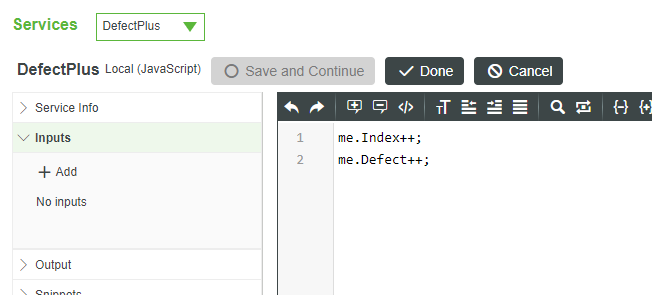


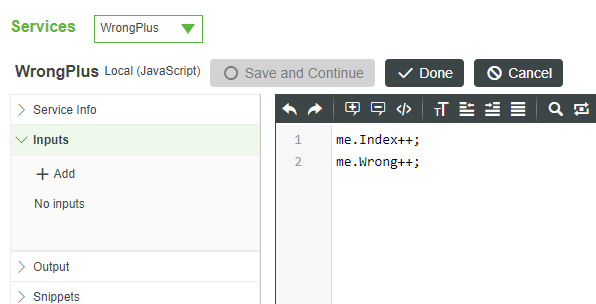
Далее, подписку, которая будет записывать данные в таблицу, при прибавлении индекса.



Далее, создаем сервисы, для прибавления правильной сборки, брака и неверного кода. Реализуем везде прибавление индекса.



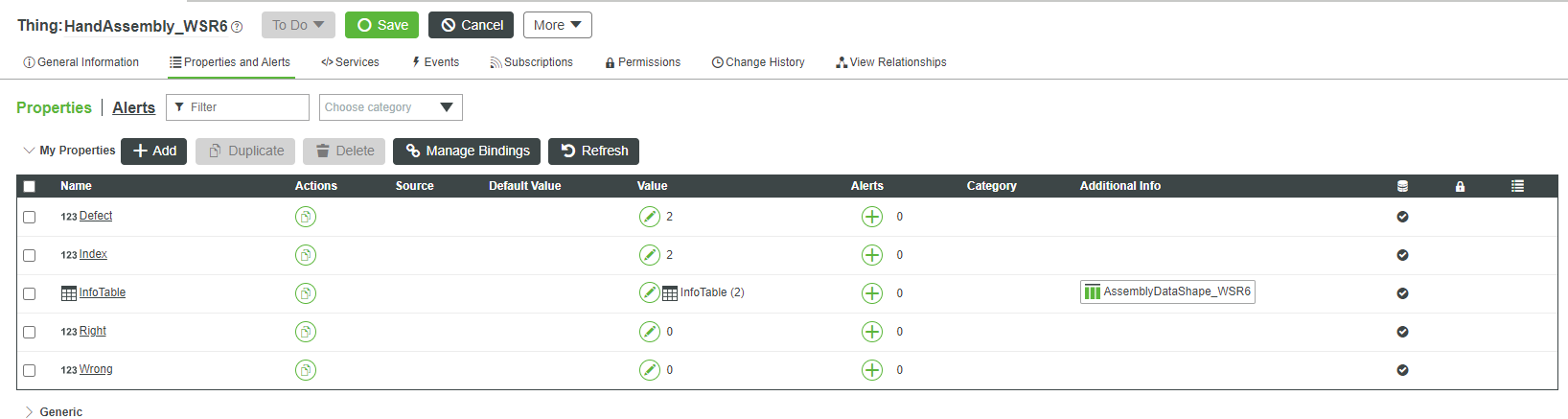




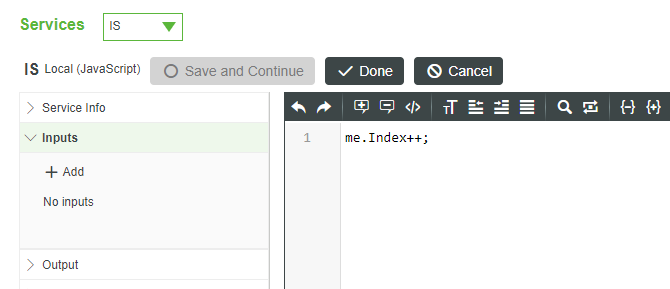
Чтобы это все работало, нужно раскидать по всей автоматизации наши сервисы.

Например, если робот выполнил команду, верно, прибавляем в коде RightPlus. Right прибавился, индекс тоже, а значит сработала подписка на запись данных в таблицу.

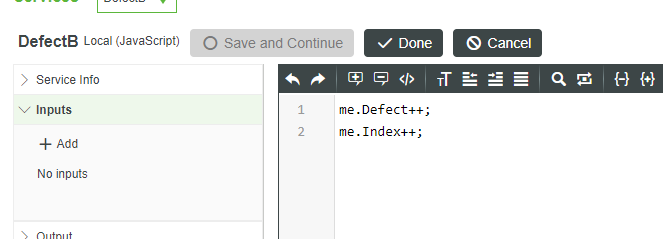
Теперь, создаем ручной учет, тут почти тоже самое:

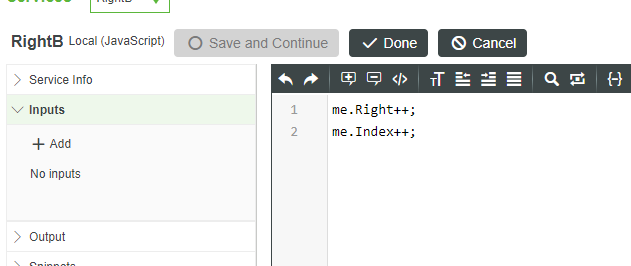


Создаем IndexService.



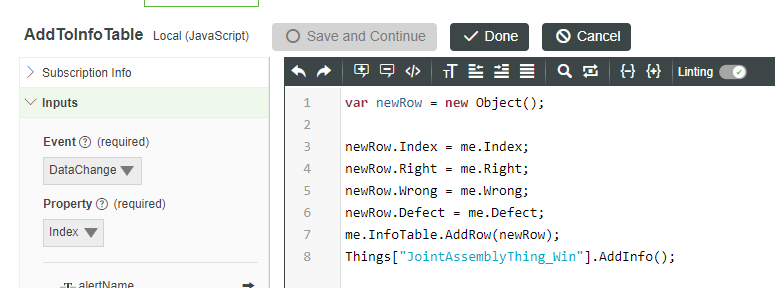
И сервисы для кнопок на мэшапе. Каждый раз, когда пользователь будет нажимать на кнопку, значения будут прибавляться и записываться.



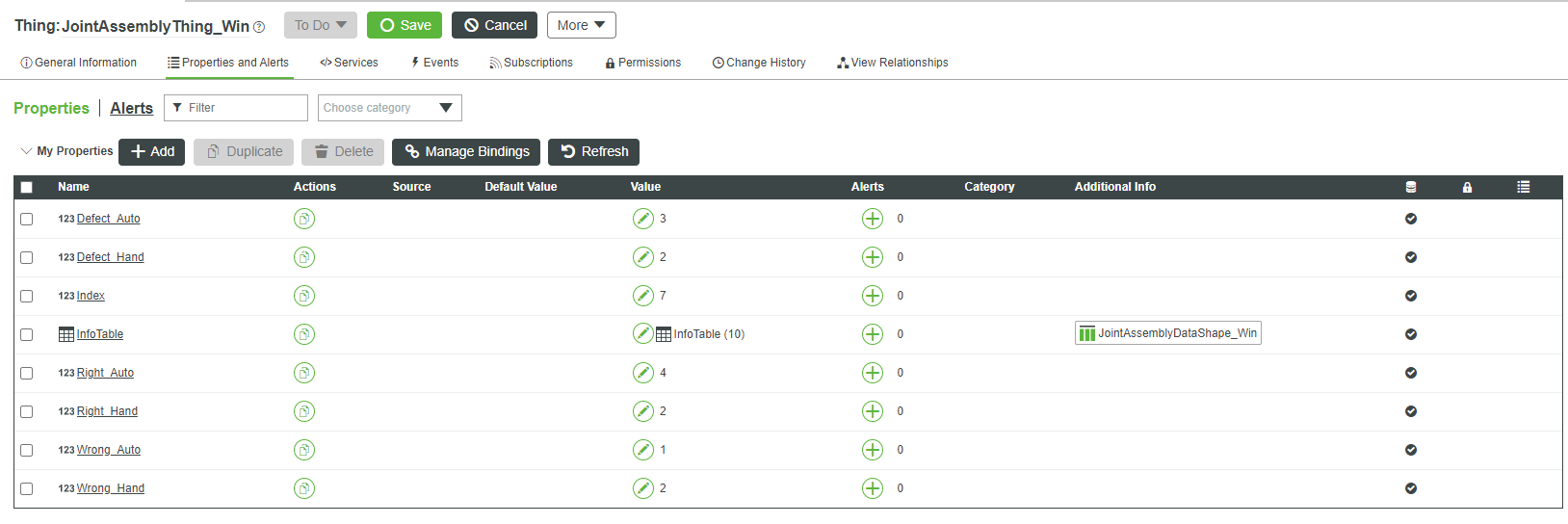




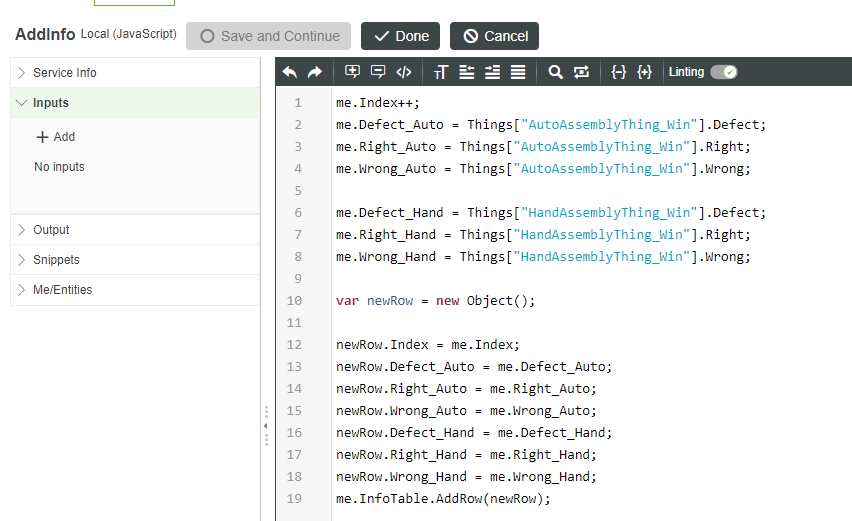
Ну и подписку, на запись данных.



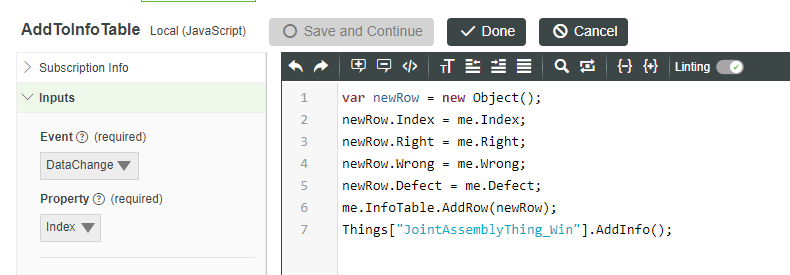
И последнее, JointAssembly.

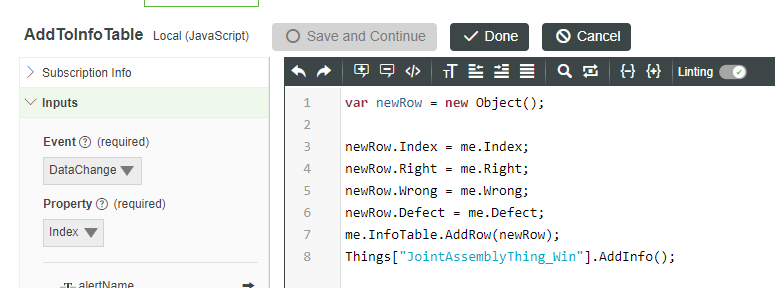


Создаем сервис «AddInfo», который будет записывать данные из созданных выше вещей.



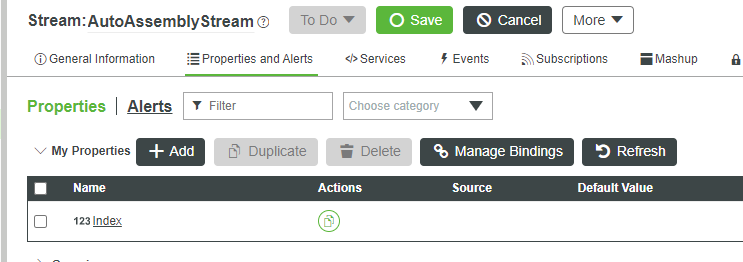
Теперь возвращаемся к старым сервисам и дополняем их. Нужно добавить выполнение сервиса AddInfo в сервисах на запись данных в ручной и автоматической сборках.



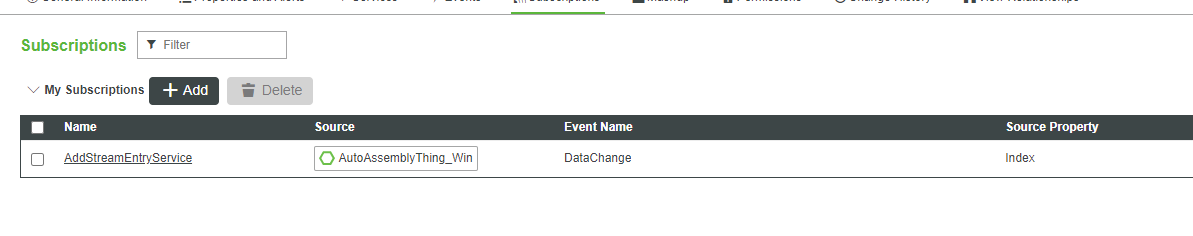


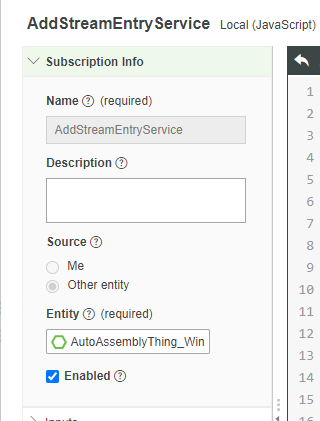
Теперь создаем Стримы для этих вещей.

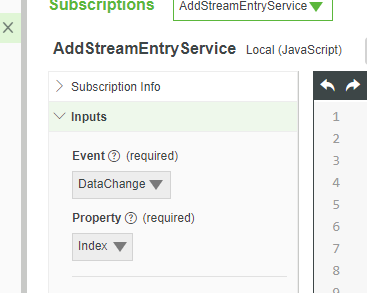
Первый стрим – Автоматические сборки. Назовем его AutoAssemblyStream



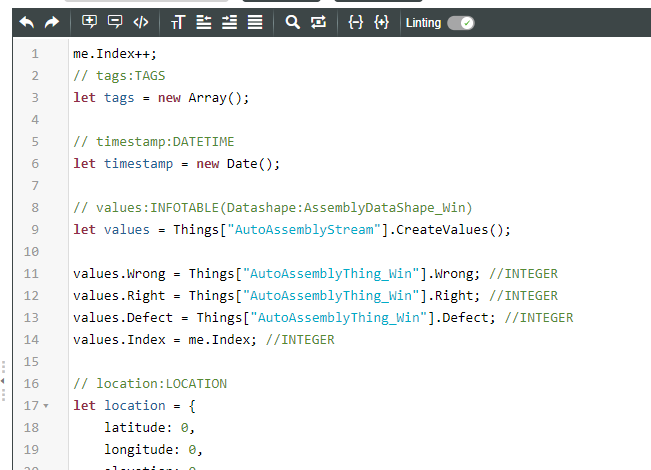
Создаем подписку, которая будет реализовывать запись данных, при изменении индекса в вещи AutoAssembly.



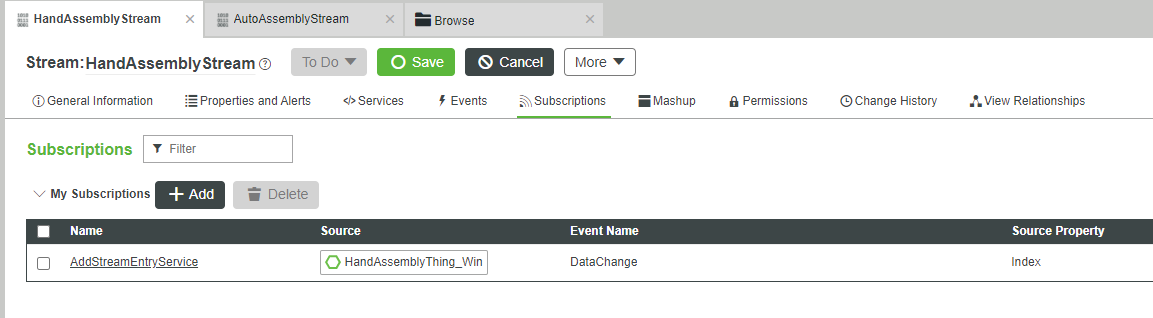


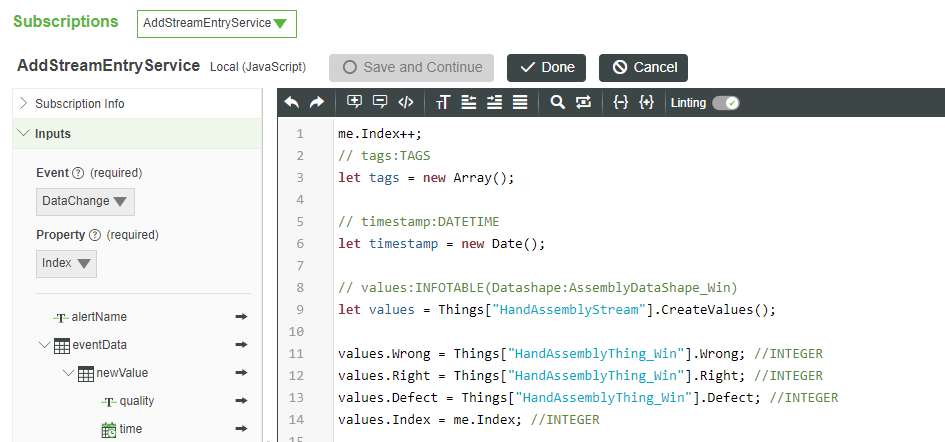


Все также, добавляем сниппет.



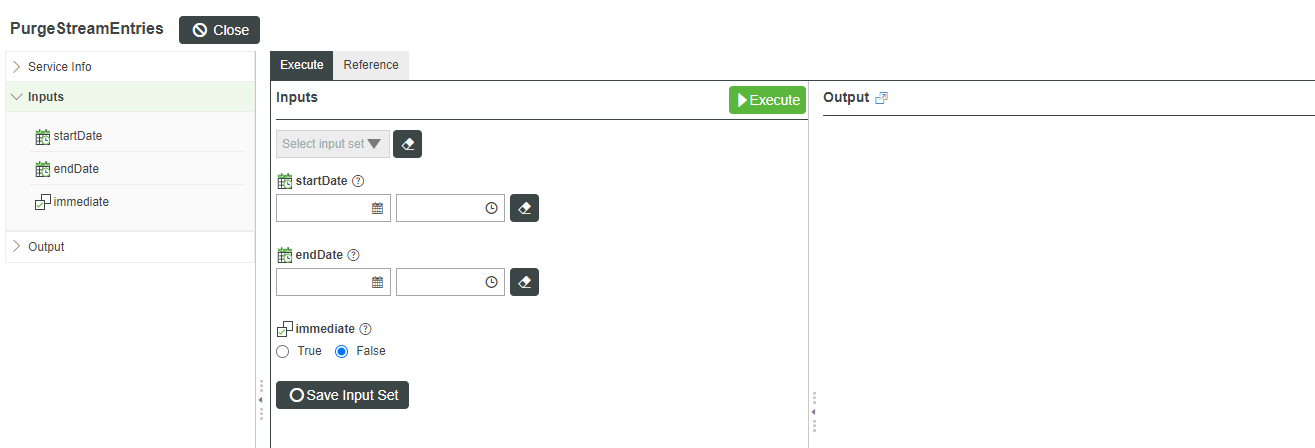
Делаем все тоже самое и для ручной сборки.





### Глава 22. Очистка логгирования

Очистка стрима происходит с помощью готового сервиса. Его можно будет привязать к кнопке на мэшапе.



### Глава 23. Описание логгируемых вещей

Выше был описан процесс создания записи данных. Точно также нужно создать еще много таких-же вещей и подписок для логгирования остальных вещей и свойств.

**Внимательно читайте задание, там все написано.**

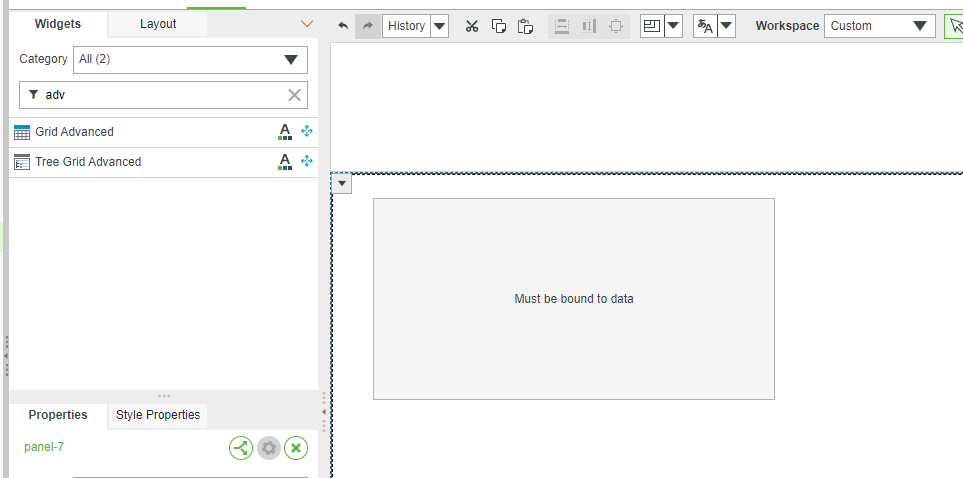
Примерный список того, что нужно записывать:

* Load (нагрузка сервомоторов);
* Motor (положение сервомоторов);
* Temp (температура сервомоторов);
* Control (позиции, статус и прочее);
* DeadManSwitch (WorkShiftTerminal);
* SmartCamera;
* Assembly (Hand;Joint;Auto).

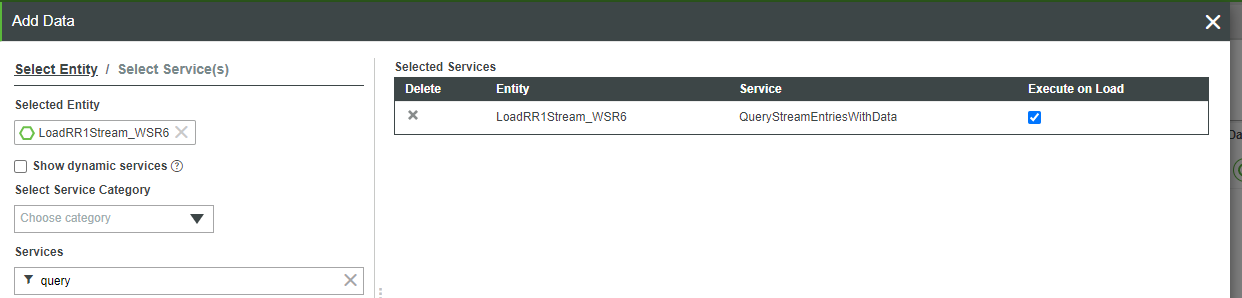
### Глава 24. Статистика на мэшапе

Статистика и визуализация необходима на конкретных мэшапах по заданию. Например на мэшапе «Руководителя» и «Контроля качества изделий».

Добавляем виджет «Advanced Grid»

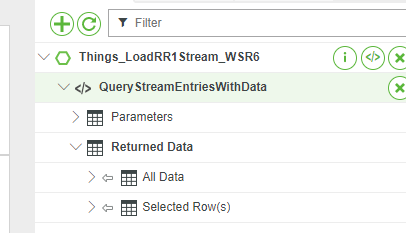


В правом верхнем углу, загружаем данные. Тут выбираем необходимую вещь, например, нагрузка сервомоторов – LoadStream.

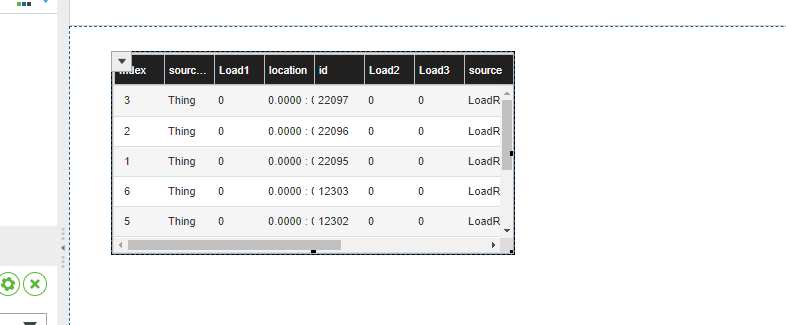


И загружаем сервис «QueryStreamEntriesWithData». Не забываем ставить галочку «Execute on Load», она заставит сервис сработать при загрузке мэшапа.

Из данных, которые мы загрузили находим «All Data» и перетаскиваем на наш созданный грид.



В итоге, данные загрузились.



Тоже самое и с графиком. Выбираем необходимый для вас график и переносим его на мэшап. Перемещаем на него данные и график будет работать.

