TQL

Language Specification & Kickstart Guide

Thing Definition (TDL), Workflow Definition (WDL), Thing Interaction/Query (TIL)

Version 1.0

Table of Contents

1. Introduction 1

2. TQL Hierarcy 2

2.1 Domain-Specific Language Background 2
2.2 Hello world 3
2.3 Atomic Language Constructs 4
2.3.1 Handle to Request and Response Objects 4
2.3.2 Adding custom Header values to a HTTP Response 5
2.3.3 Sepcial Variable Storage and Scoping 6
2.3.4 Selection Statements 7
2.3.4.1 The if Statement 7
2.3.4.2 Select / Case Statements 7
2.3.5 Process Commands 8
2.3.6 DoRequest with specified target 9
2.3.7 Iteration statements 9
2.3.7.1 The for each statement 9
2.4 Thing Definition Language (TDL) 10
2.4.1 List of Primitive Attribute types 13
2.5 Workflow Defintion Lanauge 13
2.6 Thing Interaction Lanauge (TQL) 15
2.6.1 Find a model with Single Condition 17
2.6.2 Find model with multiple conditions 18
2.6.3 Find single row using specific ID value 19
2.6.4 Find all the data in a given Model (Find all rows) 19
2.6.5 Update one singe row 20
2.6.6 Update Multiple Rows 20
2.6.7 21
2.6.8 21
2.6.9 21

3. Creating Thing Facets 23

3.1.1 TQLEngine Protocol Handler support Matrix. 23

4. OnBoarding of Devices (Sensors, etc) 25

5. TQLEngine Sandbox, Local Environment 26

5.1 TQLEngine 26
5.2 Deploying locally on your environemnt 26
5.2.1 Sandbox Environment 27
5.2.2 Checking Health, GetFacetInstance from TQLEngine. 27
5.2.3 How to deploy a FacetScript to TQLEngine 28

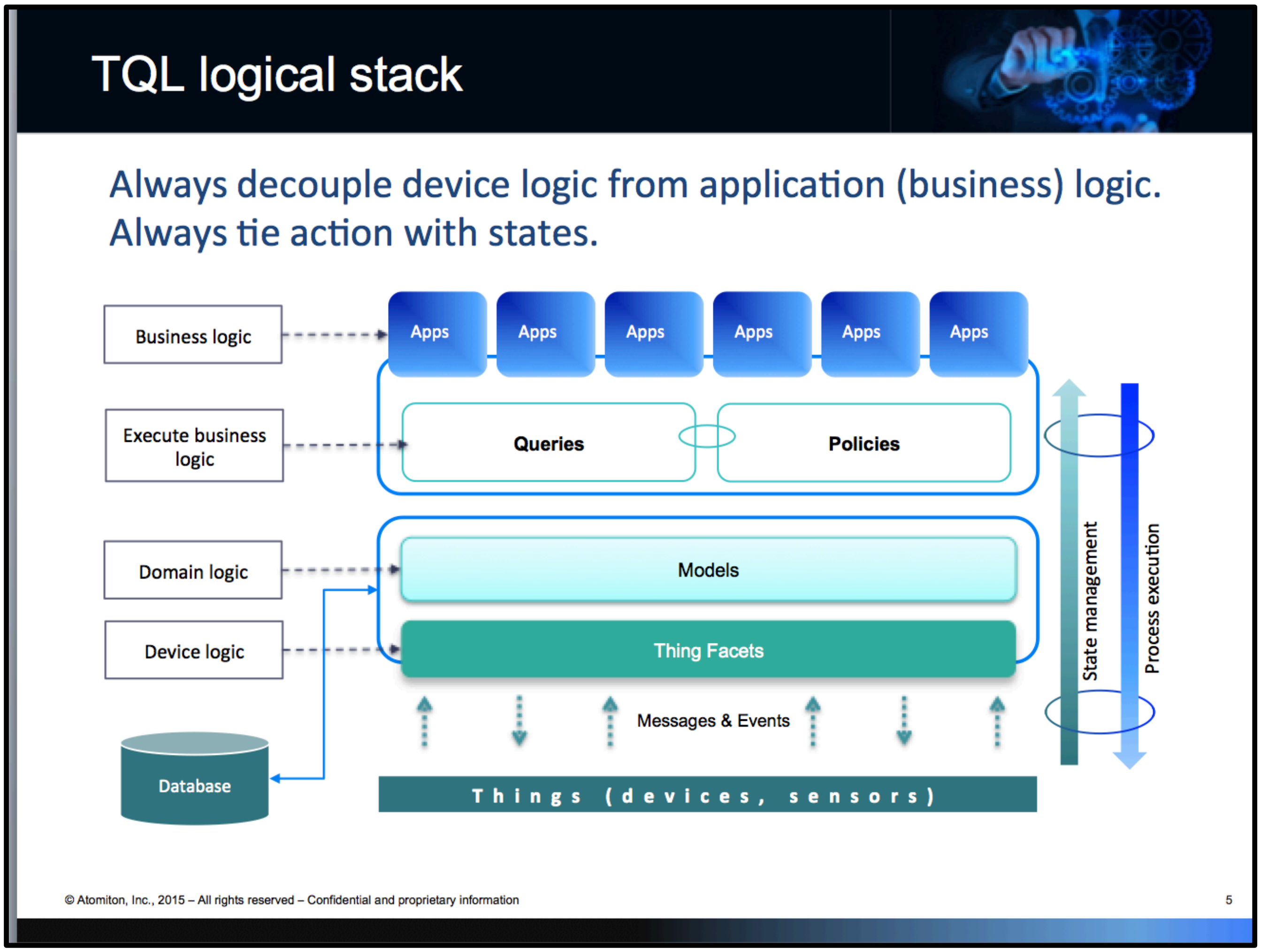
A. Practical Examples 33

A.1 How to enable access to static files in your local filesystem over an HTTP Transport 33
A.2 How to create a simple ThingFacet Script to enable Thing Queries (TQL) over an HTTP Transport 33
The simplest way to enable TQL Queries over an HTTP transport is to deploy a new facet instance with SffTqlFacet type. In the OnActivation flow you can specify the reference to the data/thing model using <Include> tag. 33
Example: 33
</NewPackage> 35
A.3 How to start a WebSocket Server 35

A.4 An example : Smart Parking 35
A.4.1 SmartParking Data Model File (ParkingLotAssetsTQL.dbm.xml) 35
A.4.2 FacetScript source code.. 40

1. Introduction

TQL is an IoT Application Platform that creates queriable and actionable systems of interconnected things. Using TQL, IoT application developers can easily implement interactive and responsive IoT business logic using TQL statements.



TQL logical stack

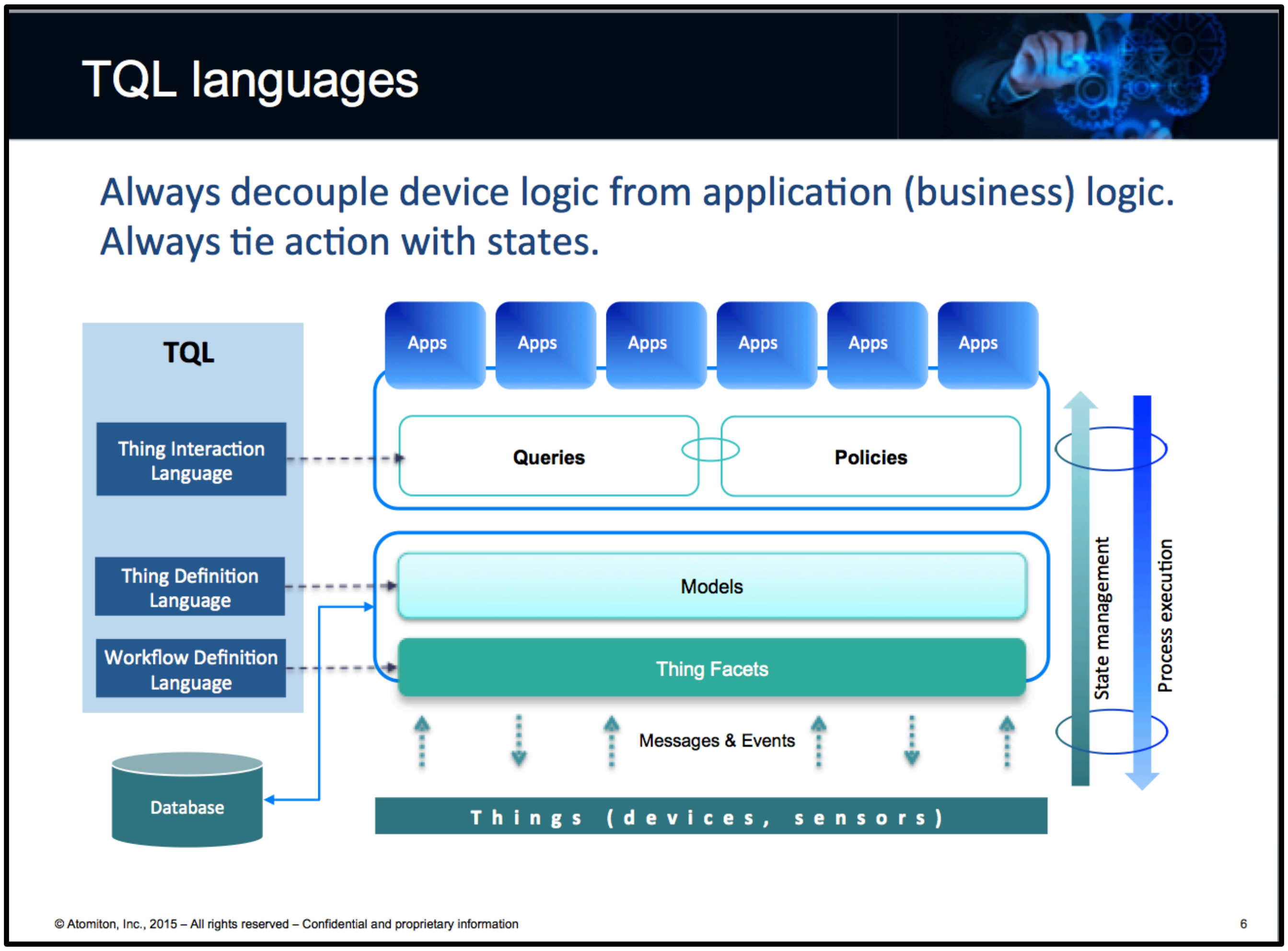
Always decouple device logic from application (business) logic.
Always tie action with states.

© Atomiton, Inc., 2015 – All rights reserved – Confidential and proprietary information

5

2. TQL Hierarcy

Thing Definition (TDL), Thing Interaction (TIL) and Workflow Definition (WDL) are part of Atomiton's Atomic domain languages targeted to be a single unified language for interaction with broad range (Billions) of machines (Devices) that may be isolated, inter-connected, each with differernt message formats and communication protcols.


TQL languages
Always decouple device logic from application (business) logic.
Always tie action with states.
TQL
Thing Interaction Language
Thing Definition Language
Workflow Definition Language
Database
Apps
Apps
Apps
Apps
Apps
Apps
Queries
Policies
Models
Thing Facets
Messages & Events
Things (devices, sensors)
State management
Process execution
© Atomiton, Inc., 2015 – All rights reserved – Confidential and proprietary information
6

2.1 Domain-Specific Language Background

A domain-specific language is a computer language specialized to a particular application domain. This is in contrast to general-purpose language (GPL), which is broadly applicable across domains, and lacks specialized features for a particular domain. There is a wide variety of DSLs, ranging from widely used languages for common domains, such as HTML for web pages, down to languages used by only a single piece of software. DSLs can be further subdivided by the kind of language, and include domain-specific *markup* languages, domain-specific *modeling languages* (more generally, specification languages), and domain-specific *programming* languages. Special-purpose computer languages have always existed in the computer age, but the term "domain-specific language" has become more popular due to the rise of domain-specific modeling. [***Source***: http://en.wikipedia.org/wiki/Domain-specific_language]

A **modeling language** is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure

Domain-specific modeling (DSM) is a software engineering methodology for designing and developing systems, most often IT systems such as computer software. It involves systematic use of a graphical domain-specific language (DSL) to represent the **various facets** of a system. DSM languages tend to support higher-level abstractions than General-purpose modeling languages, so they require less effort and fewer low-level details to specify a given system. [***Source***: http://en.wikipedia.org/wiki/Domain-specific_modeling]

2.2 Hello world

The "Hello, World" program is traditionally used to introduce a programming language. Here it is in FacetScript:

```
<NewPackage>
   <NewFacetInstance fid="helloworld" name="HelloWorld"
      type="SffTcpFacet">
      <OnActivate />
      <OnOpen ModifyPipeline="HttpServerExtensionArgs" />
      <OnRequest>
         <DoResponse>
            <Process>
               <Message type="text">
                  <Value>
                     Hello World from FacetScript!
                  </Value>
               </Message>
            </Process>
         </DoResponse>
      </OnRequest>
      <OnResponse />
      <OnClose />
   </NewFacetInstance>
</NewPackage>
```

FacetScripts source files typically have the file extension `. xml`. Assuming that the "HelloWorld" facetscript is stored in the file `helloworld.xml` and deployed to the TQLEngine server, (Please note that this document does not provide details on how to deploy facetscripts to the TQLEngine) the facetscript can be executed using the HTTP Request as follows:

```
http://localhost:8080/fid-helloworld
```

which produces HTTP Response as

```
Hello World from FacetScript!
```

The “Hello, World” facetscript starts with NewPackage which contains the definition of deployable unit to TQLEngine. A package contains - NewFacetInstance, a special kind of tag that contains message processing logic based on request / response from external entities.

The `HelloWorld` facetinstance declared by the “Hello, World” FacetScript has a processing function only in the DoRequest phase. NewFacetInstance declaration takes three attributes:

- ***fid:*** *This is the unique secured id that is assigned for the facet instance. This will be autogenerated when ThingFacets are constructed using Atomiton’s TQLStudio.*
- ***name:*** *This give a name to the NewFacetInstance.*
- ***type:*** *This specifies the FacetType to be used. In most cases this will be a plain Message Facet Type i.e. SffTcpFacet.*

There are five phases of execution that the user has control over to write their custom processing logic. They are:

- ***OnActivate:*** *This phase is executed only once during the deployed of the FacetInstance to the TQLEngine.*
- ***OnOpen:*** *This phase is executed everytime a connection is opened from an external source (human (or) other FacetScripts). OnOpen takes an attribute ModifyPipeline to provide the external transport protocol. The values are: HttpServerExtentionArgs (For HTTP-based external transport) and WsServerExtensionArgs (For Websocket-based external transport).*
- ***OnRequest:*** *This is the main phase of execution of the request. Users can do bulk of their processing logic during this phase.*
- ***OnResponse:*** *This phase is executed when the response is about to leave to the requester.*
- ***OnClose:*** *This phase is executed when the external connection is being closed after serving the request.*

2.3 Atomic Language Constructs

Atomic Language constructs pertains to the basic elements, commands, and statements using in various subdomain TQLs – TDL, TIL and WDL.

2.3.1 Handle to Request and Response Objects

Within a NewFacetInstance (OnOpen, OnRequest, OnResponse, OnClose) you can get access to Request object by using notation $Request.

For example if the Request to the NewFacetInstance is: http://localhost:8080/fid-helloworld/test/uri?test=1

You can drill down to each element:

$Request.URI

$Request.Arguments

$Request.Arguments.Test (Access to Test Argument value): Returns 1 for example above.

$**Request.Message.Value** (Access to entire POST Body): Predefined Tags by the Lanaguage: <Message></Message>

$Request.Headers.host (Access to host header parameter value)

```
<Method>GET</Method>
<URI>test/uri</URI> ←---- URI (Following fid-helloworld and
before ?)
<Arguments>
    <test>1</test> ←---- Arguments that are part of a string
after ?
</Arguments>
<Cookies/>
<Headers>
    <host>localhost:8080</host>
    <connection>keep-alive</connection>
    <cache-control>no-cache</cache-control>
    <user-agent>Mozilla/5.0 (Macintosh; Intel Mac OS X
10_10_0) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/41.0.2272.76 Safari/537.36</user-agent>
    <accept>*/*</accept>
    <accept-encoding>gzip, deflate, sdch</accept-encoding>
    <accept-language>en-US,en;q=0.8</accept-language>
</Headers>
<Message Type="text">
    <Value/>
</Message>
<Protocol>http</Protocol>
<ProtocolVersion>HTTP/1.1</ProtocolVersion>
```

Similar to $Request you can get access to $Response object as well.

2.3.2 Adding custom Header values to a HTTP Response

Let's say you want to insert custom header values as part of HTTP Response. This can be achieved as part of <DoResponse> Tag.

First define a custom tag called <RESTHeaders> and within that use <Headers> required tag.

```
<RESTHeaders>
     <Headers>
       <Connection>close</Connection>
       <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>
```

```
                <Access-Control-Allow-Methods>GET, POST, PUT
                </Access-Control-Allow-Methods>
                <Access-Control-Allow-Headers>Content-Type
                </Access-Control-Allow-Headers>
            </Headers>
    </RESTHeaders>
```

Once the custom tag is defined, you can include it as part of DoResponse, Process tag as follows:

```
            <DoResponse>
                <Process>
                    <Include>RESTHeaders</Include>

                </Process>

            </DoResponse>
```

Example:

2.3.3 Sepcial Variable Storage and Scoping

Facet Script provides access to three special type of variable store. They are – ContextData, FacetData and LocalData. Each of the variable store comes with its own scoping and read/write properties.

Special type of variable storage are important handling variables across different program segments.

	Scope	**Read/Write**
FacetData	FacetInstance	Read-Only
ContextData	OnRequest	Read/Write
LocalData	Within local processing blocks like – <process>, <if>, <case>, <for each>	Read/Write

You can set the context data using following script tag

```
<SetContextData key="qexec" value="false" />
In order to acces the data do - $ContextData.qexec
```

You can delete the context data using

```
<DelContextData key="qexec"/>
```

You can set the local data using following scrip tag

```
<SetLocalData key="keyName" value="keyValue" />
In order to acces the data do - $LocalData.qexec
```

2.3.4 Selection Statements

Selection statements select one of a number of possible statements for execution based on the value of some expression.

selection-statement:
if-statement
switch-statement

2.3.4.1 The if Statement

The `if` statement selects a statement for execution based on the value of a boolean expression.

if-then statement:
`<if>` `(` *condition="boolean-expression"* `)` `<then>` *embedded-statement* `</if>`

`<if>` `(` *condition="boolean-expression"* `)` `< then>` *embedded-statement* `<else>` *embedded-statement* `</else>`

`Example:`

```
<if condition="$Request.Arguments/Model = 'magneticSensor'">
                <then>
                    <process>
                    </process>
                </then>
</if>
```

An if statement is executed as follows:

- The boolean expression is evaluataed as specified in condition=
- If the boolean expession yields true, control is transferred to the first embedded statement. When and if control reaches the end point of that statement, control is transferred to the end point of the if statement.
- If the boolean expression yields false and if an else part if present, control is tranferred to the second embedded statement. When and if control reaches the end point of that statement, control is transferred to the end point of the if statement.
- If the boolean expression yields false and if an else part if not present, control is transferred to the end point of the if statement.
- Nested if statements are supported as well.

2.3.4.2 Select / Case Statements

The select / case statement selects for execution a statement list having an assoiciated case label that corresponds to the value of the select expression.

select-statement:

<select when=(expression) /> case-block

case-block:

<case when="value">

```
        <process>
        </process>
    </case>
default-block:
    <default>
        <proess>
        </process>
    </default>
```

A select-statement consists of the tag select, followed by a when= expression, followed by a case-block. The case-block consists of zero or more case-when blocks. Each case-section consists of one or more case-when labels followed by a statement-list.

The governing type of a select statement is established by the when=expression. If the type of the select when expression is int, float, string, then that is the governing type of the select statement.

There can be at most one default tag in a switch statement.

A select statement is executed as follows:

- The select expression is evaluated and converted to the governing type.
- If one of the constants specified in a case label in the select statement is equal to the value of the select expression, control is transferred to the statement list following the matched case label.

Example:

```
<select by="[:$Request.Arguments.Command:]">
           <default/>
           <case when="rfid">
              <Process>

              </Process>
              <DoResponse>

              </DoResponse>
           </case>
<select>
```

2.3.5 Process Commands

Process tag is the main interface within which statements can be executed. Process tag can contain variable declarations, making a request to another facetscript / target etc.

Example: Process tag does – Log, Include, Declare Variables

```
<Process>
   <Log>Response is : $Response</Log>
   <Include>RESTHeaders</Include>
```

```
        <GenMsgArgs>
           <Include>$Response.Message.Value</Include>
        </GenMsgArgs>
        <Include>GenerateMessage</Include>
     </Process>
```

Example process statement as part of DoResponse setting message / Value

```
<DoResponse >
   <Process>
      <Now>[:[:@Request:]$Now(%d):]</Now>
      <Message type="text">
         <Value>
            <Include>SetDigitalSignMessage</Include>
         </Value>
      </Message>
   </Process>
</DoResponse>
```

2.3.6 DoRequest with specified target

FacetScript allows you to make a request to different target using the target attribute as part of DoRequest.

For example:

```
<DoRequest target="ParkingLotDB">
   <Process>
      <Message type="json">
         <Value>
            <!—Message format needed by ParkingLotDB -->
         </Value>
      </Message>
   </Process>
</DoRequest>
```

In this example a request is being made to target “ParkingLotDB” which itself maybe a NewFacetInstance.

2.3.7 Iteration statements

Iteration statements repeatedly execute an embedded statement.

iteration-statement:

for-statement

2.3.7.1 The for each statement

The `for each` statement enumerates the elements of a collection, executing an embedded statement for each element of the collection.

for each-statement:
`<for each=`*local-variable-type   identifier* `in =` *expression* `)` *embedded-statement*

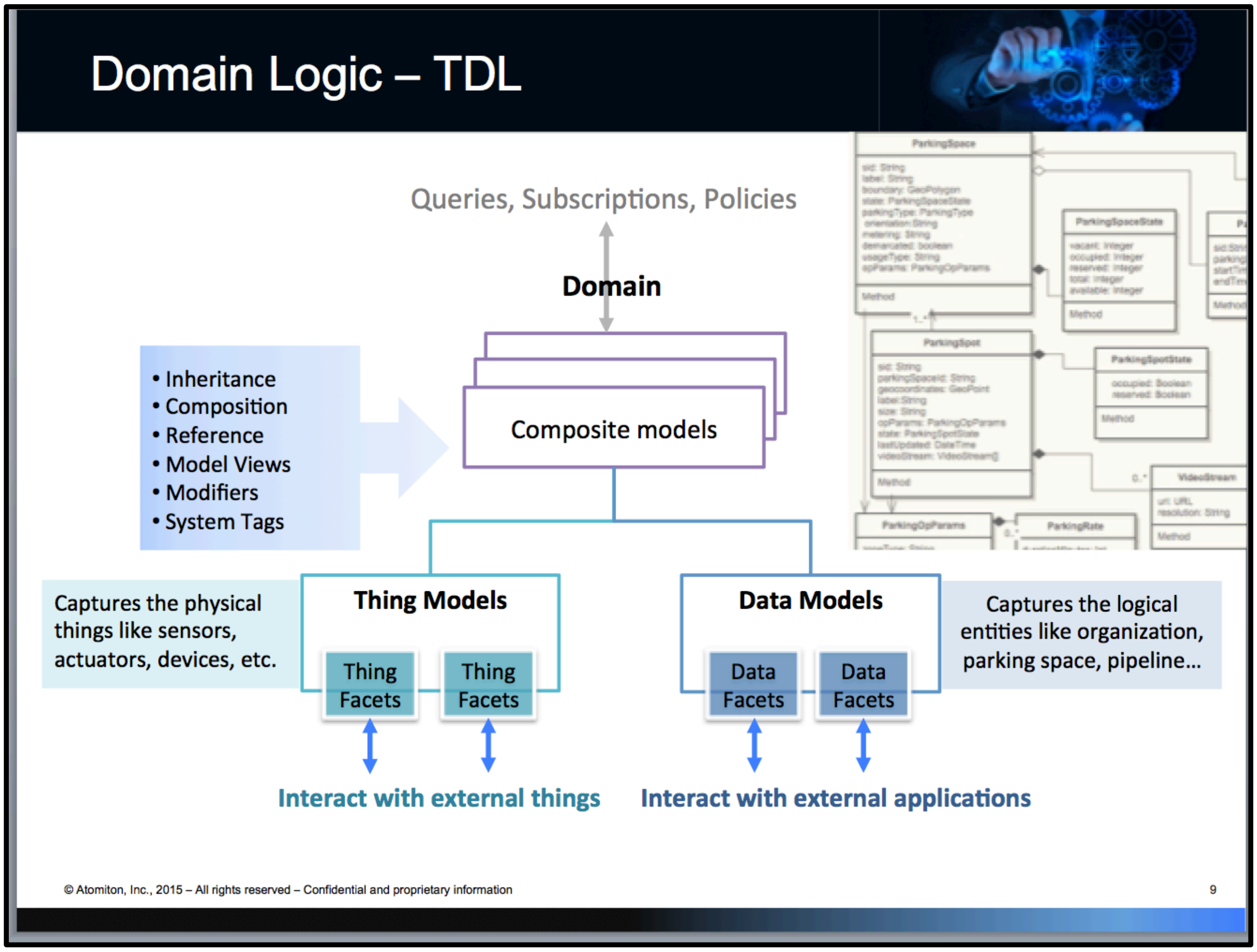
```
<for each="al" in="$Response.Message.Value.Update.AreaLight">
   <Log>Looping....$LocalData.al</Log>
   <Process>
         <!– Process something using LocalData →
   </Process>
</for>
```

In this example we iterate over an array of Objects. The *for-initializer*, consists of a *local-variable-declaration*.

The *type* and *identifier* of a `for each` statement declare the ***iteration variable*** of the statement. The iteration variable corresponds to a read-only local variable with a scope that extends over the embedded statement. During execution of a `for each` statement, the iteration variable represents the collection element for which an iteration is currently being performed.

2.4 Thing Definition Language (TDL)

The purpose of the Model definition language is to define higher level models that can be accessed from Thing Queries. Model definitions can be thought of as external interfaces to the lowel-level thing facet scripts. Model definitons are also required to store the data that is received from things or other non-things storage.


Domain Logic – TDL
Queries, Subscriptions, Policies
Domain
• Inheritance
• Composition
• Reference
• Model Views
• Modifiers
• System Tags
Composite models
Captures the physical things like sensors, actuators, devices, etc.
Thing Models
Thing Facets
Thing Facets
Data Models
Data Facets
Data Facets
Captures the logical entities like organization, parking space, pipeline…
Interact with external things
Interact with external applications
© Atomiton, Inc., 2015 – All rights reserved – Confidential and proprietary information
9

There are two types of models. A model is classified as a Thing model if it contains at least one property. A thing model can contain pure data attributes too.

Namespace: A namespace is a collection of Thing and Data Models. Namespaces provide a level of container (folder; avoid name collisions) to specific set of related thing and data models addressing a specific domain problem.

Attribute: An attribute is a specification that defines a model. It consists of name, type and a value.

Property: Properties are a special kind of attribute in which user has complete control over its getter, setter methods. In other words, the attributes are read, written using specified GET, SET operations (usually defined in a ThingFacet using Facetscript).

Operations: Operations are specialized functions or methods that does specified tasks associated with things.

Events: An event is an action or occurance detected by the program that may be handled by the program.

Brief comparison between Thing Vs Data Model

	Attribute	Property	Operations	Events	Composition	Inheritance	Reference
Data Model	**Y**	**N**	**N**	**N**	**Y**	**Y**	**Y**
Thing Model	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**

Example:

```
<Namespace name="CloverIoTApps">
   <DataModel name="CloverAppsDataModel">
      <DataFacet name="Customers">
         <Sid name="SysId" />
         <String name="firstName"/>
         <String name="lastName"/>
         <String name="title"/>
      </DataFacet>

      <ThingFacet name="IBeacon">
         <Sid name="SysId" />
         <Property name="mdid"/>
         <Double name="location"/>
         <Property name="deviceModel"/>

      <ThingFacet>

   </DataModel>

</Namespace>
```

Modeling Supports:

Inheritance:

References:

<Attribute name=”Parameter” value=”Record.Parameter”> Where Record is another Thing or Data Model

Cardinality: Defines an attribute that holds array.

Example:

<String names cardinality=”1”/> <!—Length 1→

<String names cardinality=”0..m”/> <!—Minimum size is 0 to Maximum size is m →

<String names cardinality=”1..m”/> <!—Minimums size is 1 to Maximum size of m →

2.4.1 List of Primitive Attribute types

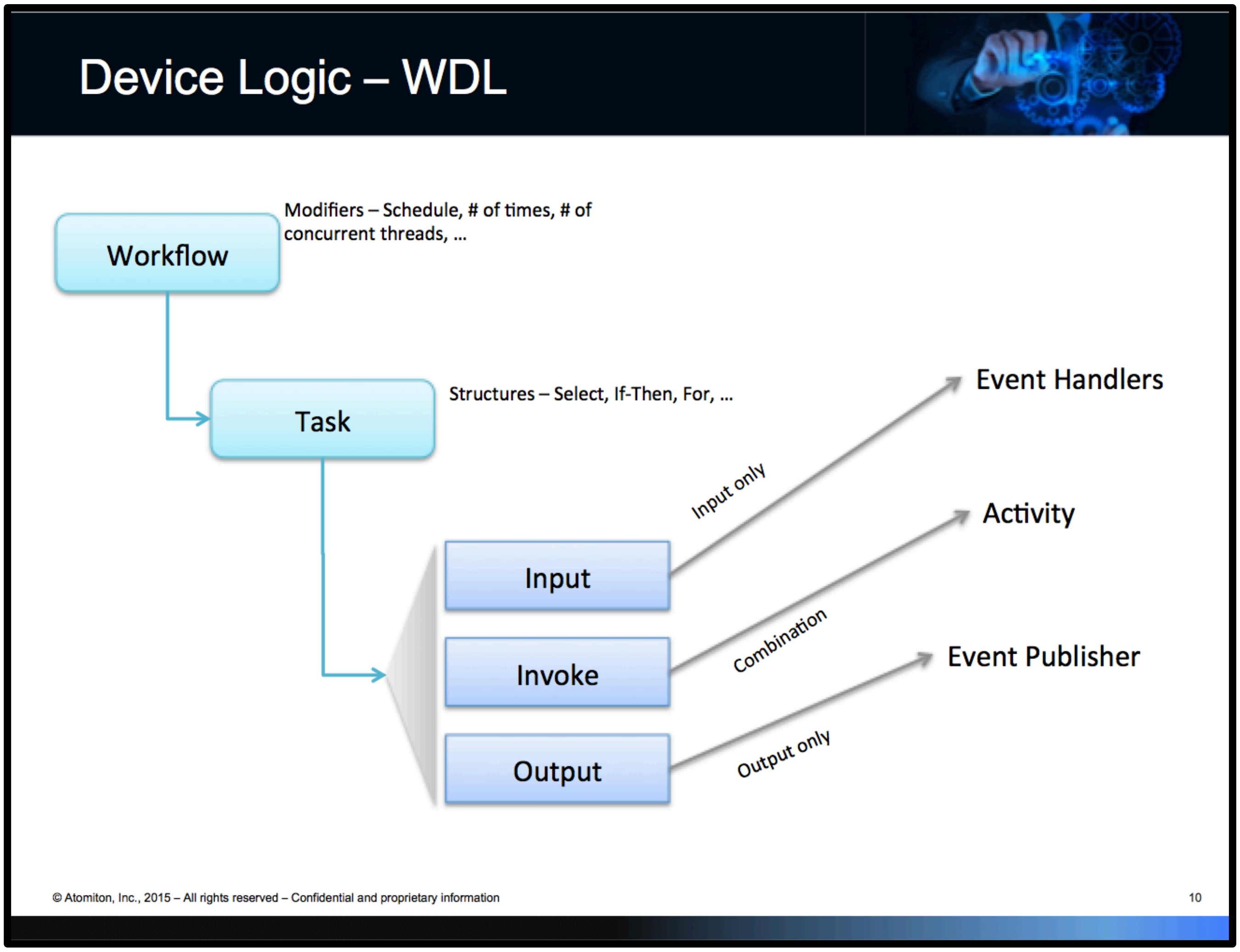
Attributes can be of any of the following primitive types

Primitive Attribute Types
String
Integer
Number
Short
Long
Boolean
Byte
Double
Float
Decimal
Datatime; with format as Java $SimpleDataFormate()
Date
Time
Duration
YearDay
MonthDay
WeekDay
YearMonth
Year
Month
Day

Sid or SystemId inherits from String type and if defined in data or thing model is automatically generated by the engine on every new row that is created.

2.5 Workflow Defintion Lanauge

In the context of Atomic language specification, Workflow definition is the automation of a device interaction and associated process (logic), in whole or part. Information or tasks are passed from one participant to another for action, according to a set of procedural rules. It consists of a numbers of logical steps, each of which is knows as an Task.

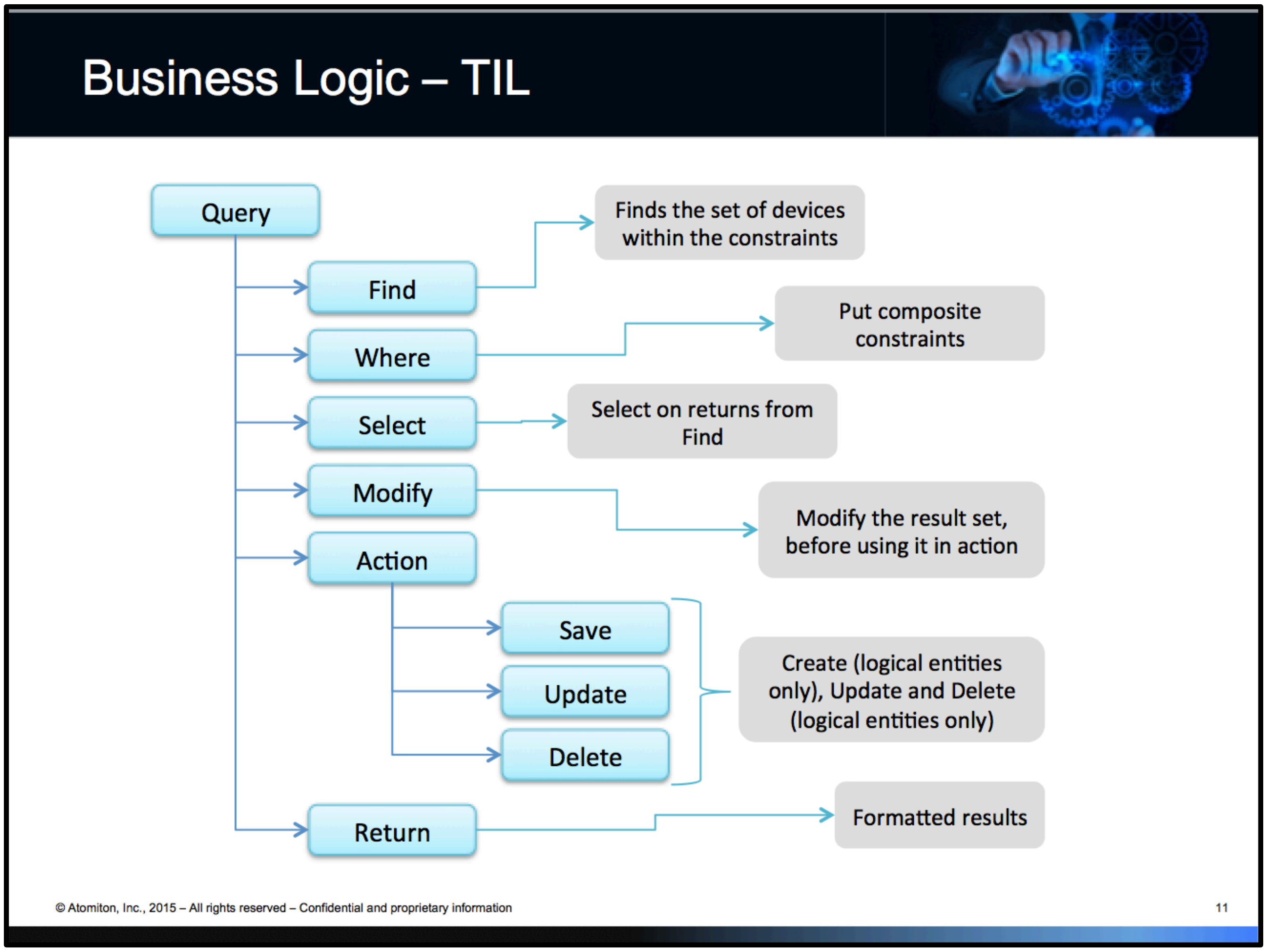


Device Logic – WDL

10

2.6 Thing Interaction Lanauge (TQL)

Thing Query Lanauge (TQL) *is yet another atomic language that is specifically designed around providing direct access for end applications to thing (or non-thing) related models and allowing users to take actions, with orthogonal applicable policices.*

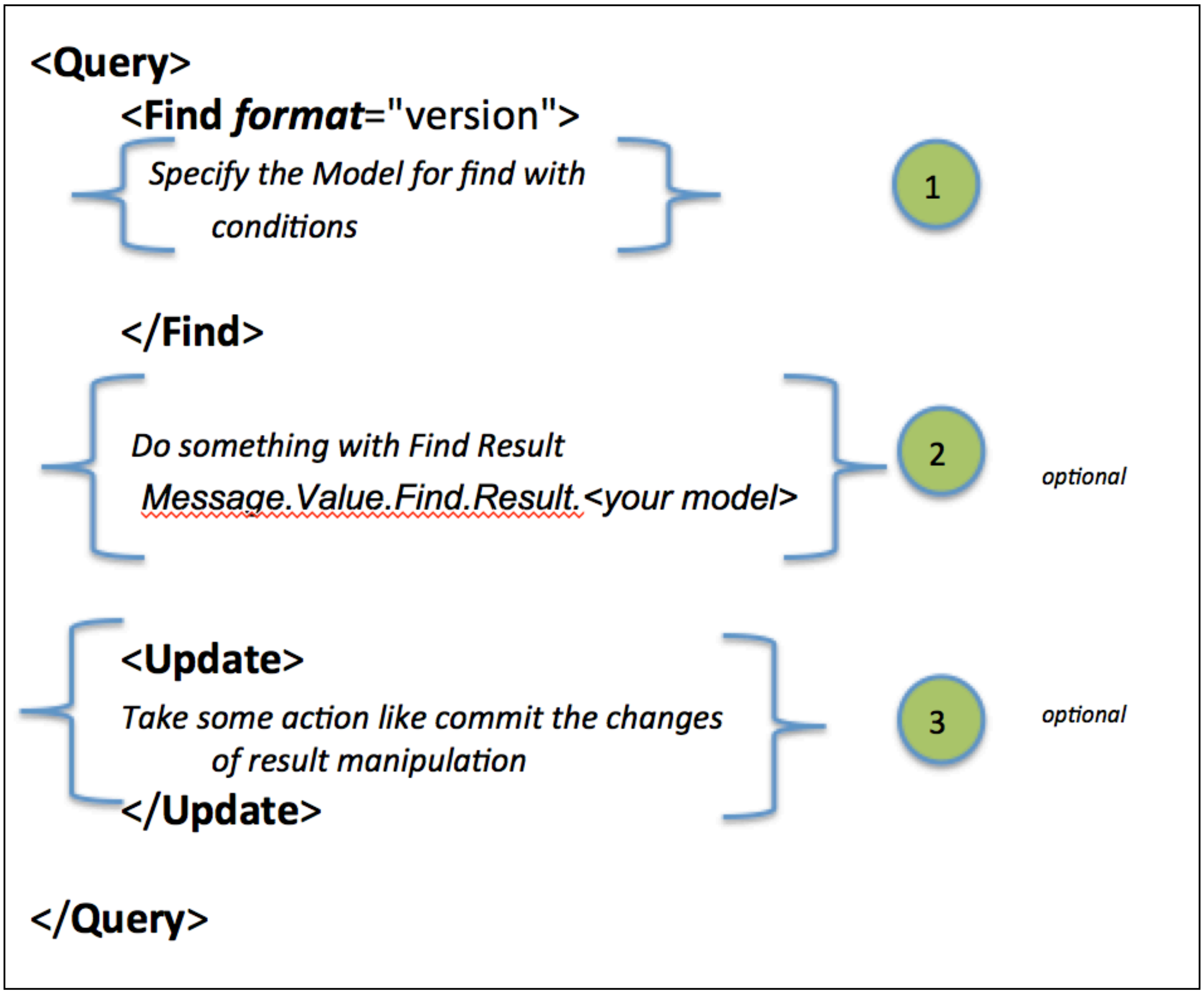

Business Logic – TIL
Query
Find
Where
Select
Modify
Action
Save
Update
Delete
Return
Finds the set of devices within the constraints
Put composite constraints
Select on returns from Find
Modify the result set, before using it in action
Create (logical entities only), Update and Delete (logical entities only)
Formatted results
© Atomiton, Inc., 2015 – All rights reserved – Confidential and proprietary information
11

The basic structure of the Thing Query is :

Step 1: Find the model

Step 2: Manipulate the Find Result set locally.

Step 3: Take action the Result i.e performing a commit i.e update back into the system, or external actions.


<Query>
<Find format="version">
Specify the Model for find with conditions
1
</Find>
Do something with Find Result
Message.Value.Find.Result.<your model>
2
optional
<Update>
Take some action like commit the changes of result manipulation
</Update>
3
optional
</Query>

Although TQL is a higher level atomic language with constrained structure, one can easily mix the FacetScript language within TQL. This comes in handy when the user wants to perform multiple updates (use for each statement).

For the examples below please refer to the Smart Parking Model and Facet Script source code section A.4

2.6.1 Find a model with Single Condition

Find Parking Spaces by their state = "on". The "on" state denotes the spot is occupied. The "off" state denotes the spot is available for parking.

The keywork Format="all" as part of Find, fetches all the internal parameters like ***order***, ***version***, ***timestamp***, ***datetime*** associcated with the model. You can filter out internal parameters from output by specifying exactly the values desired in the output. For example, format="version".

```
<Query>
   <Find format="all">
      <ParkingSpot>
         <state eq="on" />
      </ParkingSpot>
   </Find>
</Query>
```

The Result of this Query will be rows of Parking Spots.

```
<Find Format="all">
   <Result>
      <ParkingSpot sid="JPQPH22LAAAAUAABANJIGXVX"
         Type="ParkingLot.ParkingLotTQLAssets.ParkingSpot">
         <State Order="0x00" Version="1" Timestamp="1425408256843"
            DateTime="2015-03-03 13:44:16.843" Value="on" />
         <label Order="0x00" Version="1" Timestamp="1425408256843"
            DateTime="2015-03-03 13:44:16.843" Value="s" />
         <Id Order="0x00" Version="1" Timestamp="1425408256843"
DateTime="2015-03-03 13:44:16.843"
            Value="Atom-Org-1.F3.S288" />
         <parkingFloorId Order="0x00" Version="1"
Timestamp="1425408256843"
            DateTime="2015-03-03 13:44:16.843" Value="Atom-Org-
1.F3" />
         <defParkingSpotSizeId Order="0x00" Version="1"
            Timestamp="1425408256843" DateTime="2015-03-03
13:44:16.843" Value="1" />
      </ParkingSpot>
      ....
   </Result>
</Find>
```

2.6.2 Find model with multiple conditions

Find all parking spaces with state = “on” on the first floor.

```
<Query>
   <Find format="all">
      <ParkingSpot>
         <state eq="on" />
         <parkingFloorId eq="Atom-Org-1.F1" />
      </ParkingSpot>
   </Find>
</Query>
```

In this example the format of the parking floor, parking spot is :

<Organization Name>.F<floor number>.S<spot number>

Atom-Org-1.F1 -> Specifies Floor 1 managed by Atom-Org-1

2.6.3 Find single row using specific ID value

Find the parking spot information at Spot #10 on first floor. Based on the format of the ParkingSpot Id the value will be Atom-Org-1.F1.S10. Therefore the query to find 10th spot on first floor is a simple find by id as below.

```
<Query>
   <Find format="all">
      <ParkingSpot>
         <id eq="Atom-Org-1.F1.S10"/>
      </ParkingSpot>
   </Find>
</Query>
```

2.6.4 Find all the data in a given Model (Find all rows)

In order to get the entire data of a Model you can specify 1=1 type of query i.e. always return true. In the example since ID is meant to have non-null values, if we do a query on ID not equal to empty string we get back all the data!

```
<Query>
   <Find format="all">
      <AreaLight>
         <id ne="" />
      </AreaLight>
   </Find>
</Query>
```

In this example we get all the AreaLight's in the Parking Lot.

2.6.5 Update one singe row

Performing an update is a three step process – Find, Modify (locally) and Commit (Update back to the system). In this example we update the StallLight power state to "off". Stall Light is on parking spot 0 on first floor.

```
<Query>
   <Find format="version">
      <StallLight>
         <parkingSpotId>Atom-Org-1.F1.S0</parkingSpotId>
      </StallLight>
   </Find>
   <SetResponseData>
    <key>Message.Value.Find.Result.StallLight.powerState.Value</key>
    <value>off</value>
   </SetResponseData>
   <Update>
      <from>Result</from>
      <Include>$Response.Message.Value.Find</Include>
   </Update>
</Query>
```

Handle to the Result is : **Message.Value.Find.Result**.<YOUR OBJECT MODEL FLATTENED>

<SetResponseData> is a specific tag to set the Response object. Note this is a classic example of intermingling Facet Script within TQL.

2.6.6 Update Multiple Rows

In this example we perform an update to multiple rows of a Model. We are updating the AreaLight intensity to 80% on the first floor.

```
<Query>
   <Find format="version">
      <AreaLight>
         <parkingFloorId eq="Atom-Org-1.F1" />
      </AreaLight>
   </Find>
   <For each="al" in="Find.Result.AreaLight">
      <SetLocalData>
         <key>al.intensityLevel.Value</key>
         <value>80</value>
      </SetLocalData>
   </For>
   <Update>
      <from>Result</from>
      <Include>$Response.Message.Value.Find</Include>
   </Update>
</Query>
```

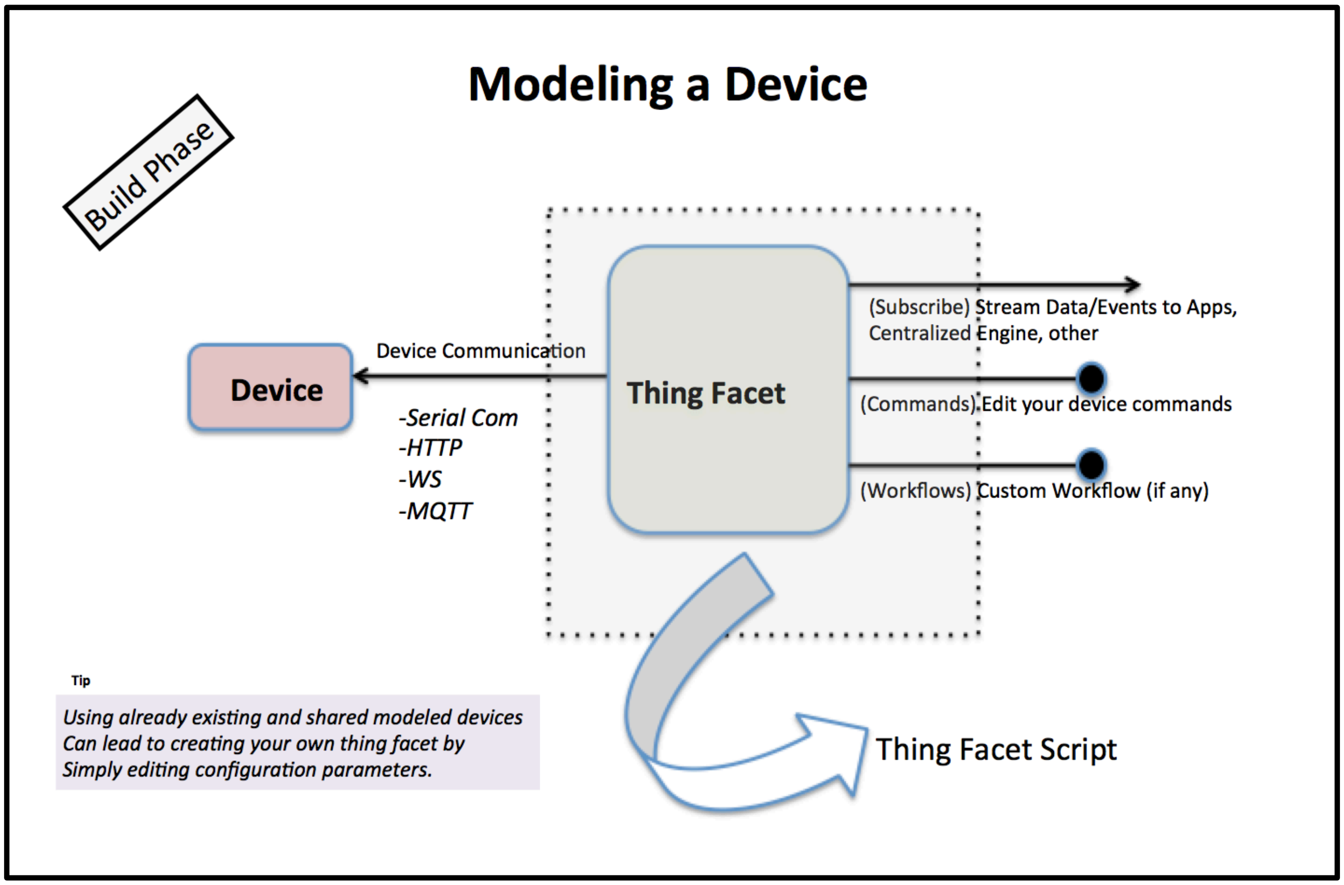
In the example we use FaceScript ***for each*** construct to loop throught the Find Result and set the descired values of the model

3. Creating Thing Facets

Thing Facets - *Capture the device logic and interaction*. Using the TQL subdomain language WDL one can easily capture the device interaction and logic.

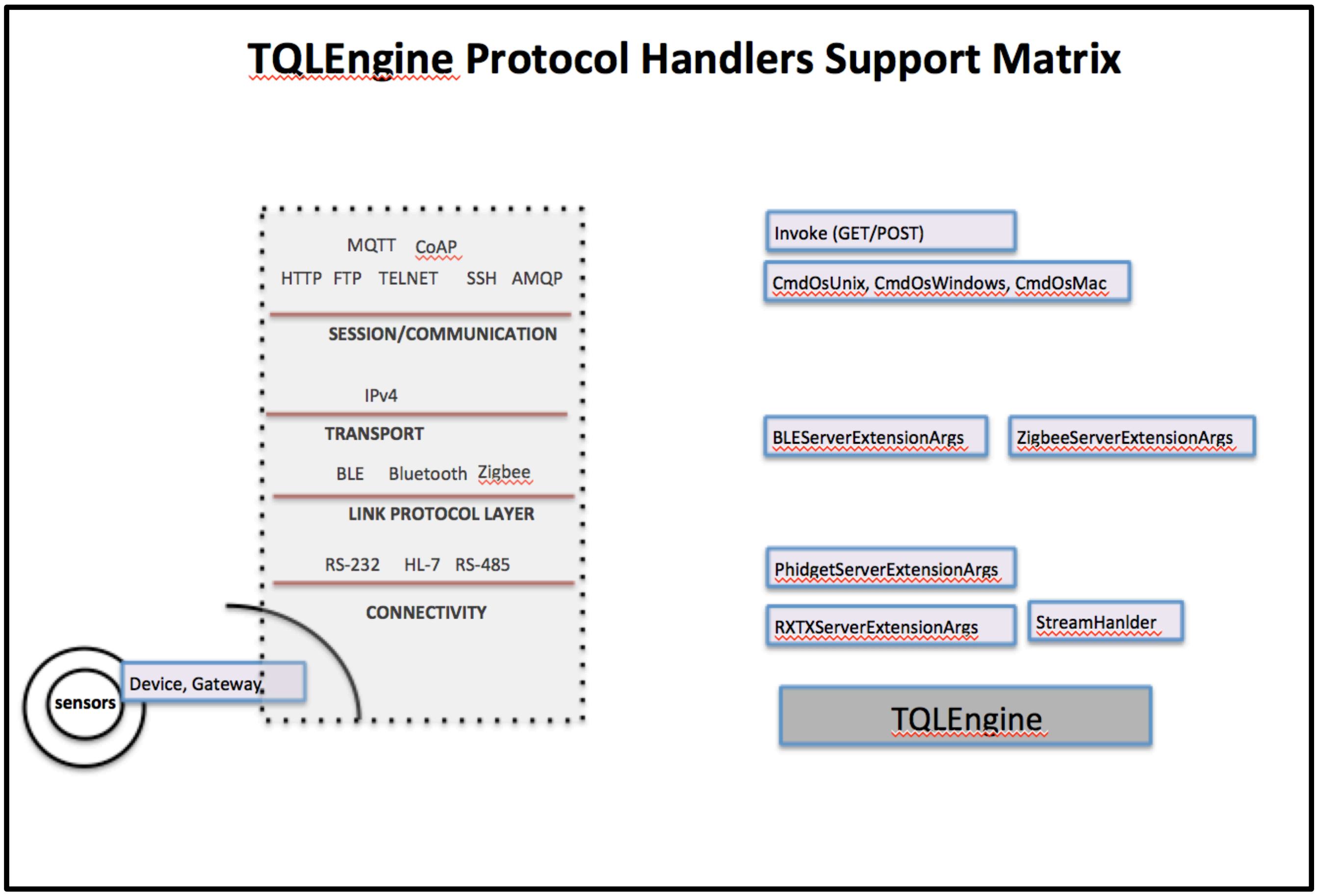
Simple steps to create Thing Facets are:

1. Use and Enable Device Communication.
2. Perform optional message transformation
3. Apply optional business logic associated with the device


Modeling a Device
Build Phase
Device
Device Communication
-Serial Com
-HTTP
-WS
-MQTT
Thing Facet
(Subscribe) Stream Data/Events to Apps, Centralized Engine, other
(Commands) Edit your device commands
(Workflows) Custom Workflow (if any)
Thing Facet Script
Tip
Using already existing and shared modeled devices Can lead to creating your own thing facet by Simply editing configuration parameters.

3.1.1 TQLEngine Protocol Handler support Matrix.

One of the key step of creating thing facet is the ability to handle the device protocol-specific communication. TQLEngine makes it simple to handle device communication by providing protocol specific parametric handlers. Users have access to protocol handlers as part of writing a thing Facet.


TQLEngine Protocol Handlers Support Matrix
MQTT CoAP
HTTP FTP TELNET SSH AMQP
SESSION/COMMUNICATION
IPv4
TRANSPORT
BLE Bluetooth Zigbee
LINK PROTOCOL LAYER
RS-232 HL-7 RS-485
CONNECTIVITY
sensors
Device, Gateway
Invoke (GET/POST)
CmdOsUnix, CmdOsWindows, CmdOsMac
BLEServerExtensionArgs
ZigbeeServerExtensionArgs
PhidgetServerExtensionArgs
RXTXServerExtensionArgs
StreamHanlder
TQLEngine

4. OnBoarding of Devices (Sensors, etc)

5. TQLEngine Sandbox, Local Environment

5.1 TQLEngine

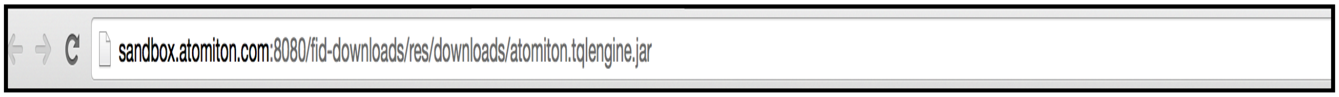
A core component of TQLSystem is the TQLEngine responsible for providing compile-time and run-time environment for Atomic Languages. TQLEngine is single deployable JAR artifact which can downloaded from the Atomiton web site.

TQLEngine Prerequisites:

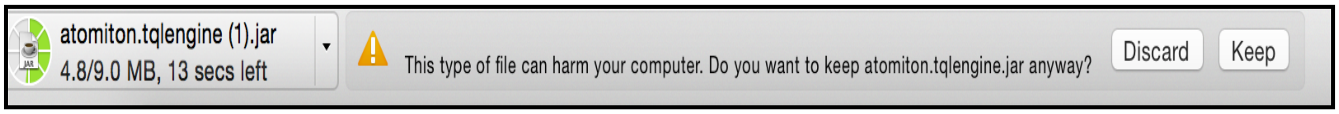
1. Linux or Windows Operating System running on any hardware – 2GB RAM, 20GB HDD, 1 CPU
2. JDK 1.7.

TQLEngine Downloads

http://sandbox.atomiton.com:8080/fid-downloads/res/downloads/atomiton.tqlengine.jar



sandbox.atomiton.com:8080/fid-downloads/res/downloads/atomiton.tqlengine.jar



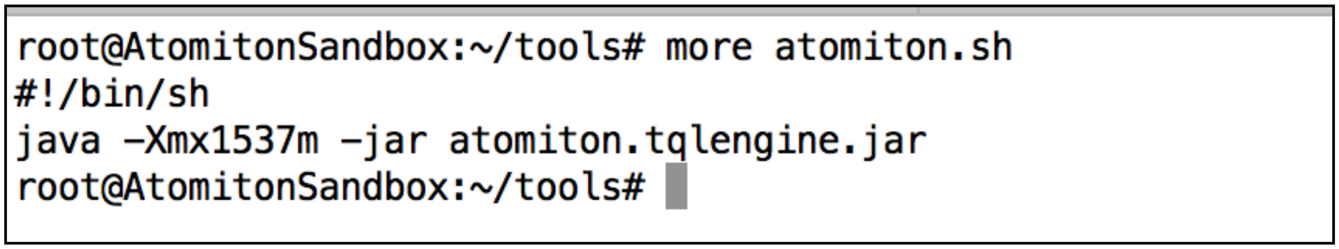
atomiton.tqlengine (1).jar
4.8/9.0 MB, 13 secs left

This type of file can harm your computer. Do you want to keep atomiton.tqlengine.jar anyway?

5.2 Deploying locally on your environemnt

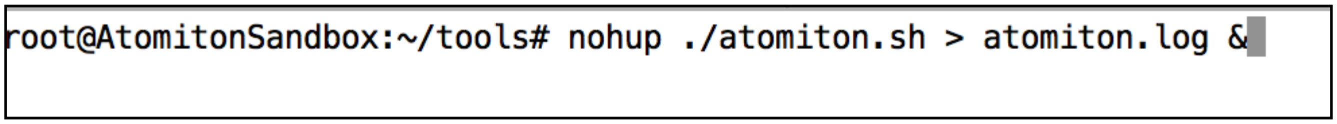
TQLEngine can be deployed locally on your environment (Mac / Windows OS). Below are the steps to setup TQLEngine locally on your development environment.

- Create a simple shell script say ‘atomiton.sh’ to start the engine. In this example we are starting the engine using Java command and using 1.5GB of memory heap. You can leave the value to default and no specify any value if you want to restrict the initial memory heap usage by TQLEngine.



```
root@AtomitonSandbox:~/tools# more atomiton.sh
#!/bin/sh
java -Xmx1537m -jar atomiton.tqlengine.jar
root@AtomitonSandbox:~/tools# 
```

- Start the script using nohup command, sending the stdout logs to a local file. In the example below we send the output to atomiton.log file



```
root@AtomitonSandbox:~/tools# nohup ./atomiton.sh > atomiton.log &
```

5.2.1 Sandbox Environment

A Shared Sandbox environment is created on Atomiton's Cloud Infrastructure.

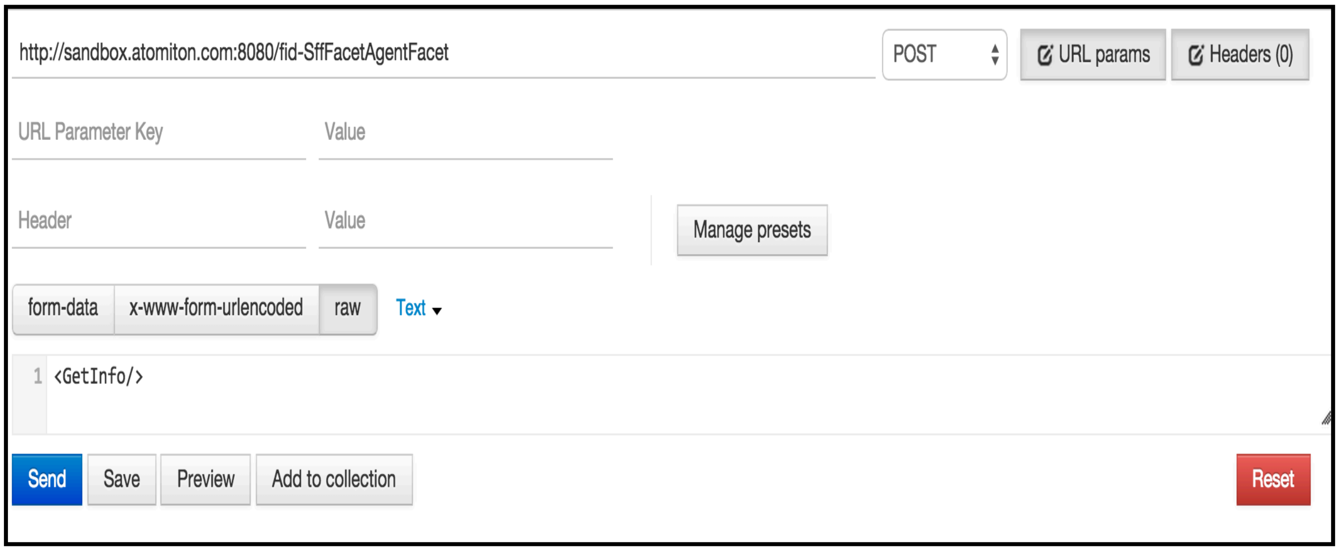
TQLEngine can be accessed from the location:

http://sandbox.atomiton.com:8080/

5.2.2 Checking Health, GetFacetInstance from TQLEngine.

The initial FacetInstance of TQLEngine is fid-SffFacetAgentFacet. This is the main target to perform health, and other Facet related information from the TQLEngine.

To Get Information about the Engine run the command <GetInfo/>



http://sandbox.atomiton.com:8080/fid-SffFacetAgentFacet

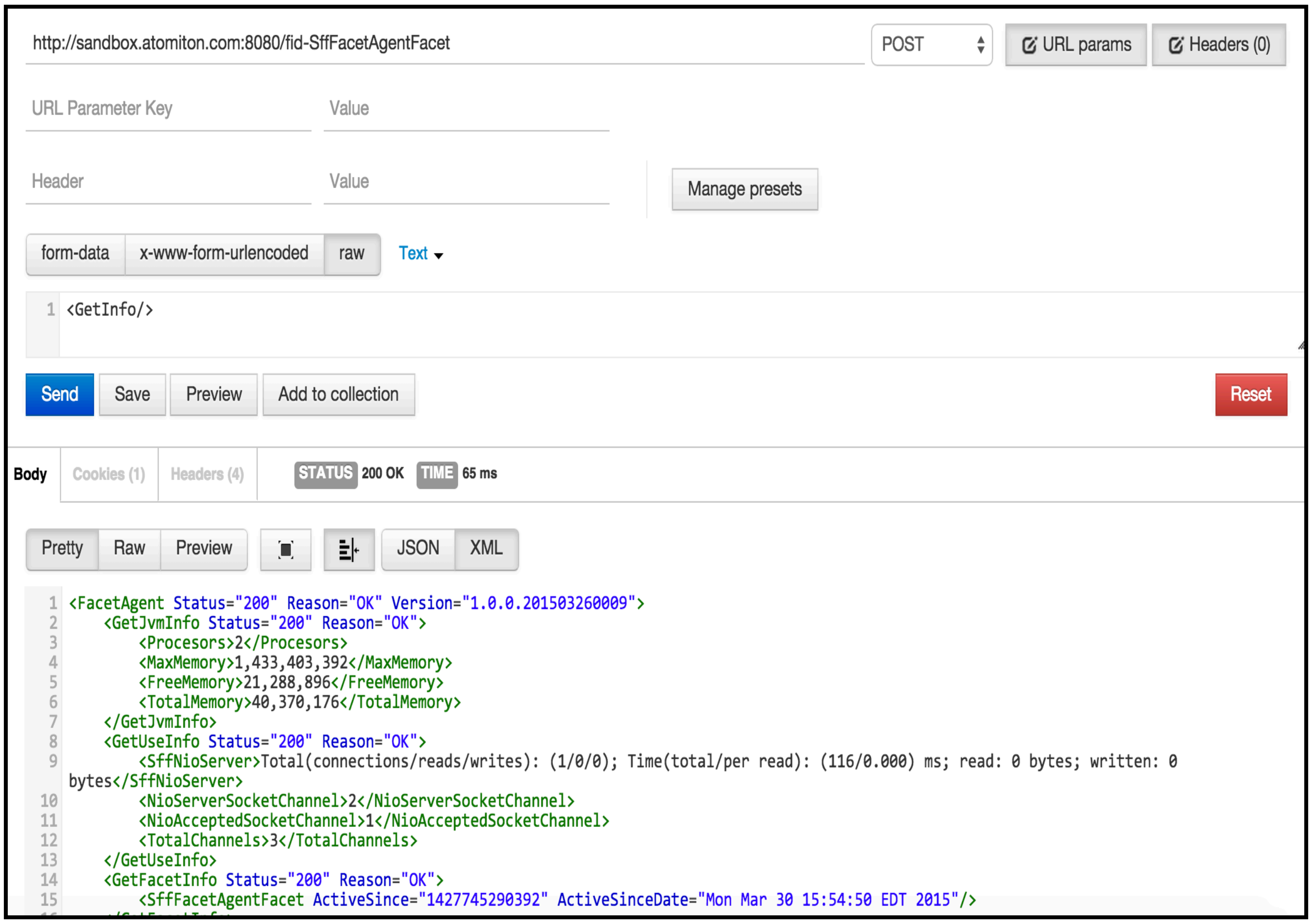
POST

```
<GetInfo/>
```

```
<FacetAgent Status="200" Reason="OK" Version="1.0.0.201503260009">
   <GetJvmInfo Status="200" Reason="OK">
      <Procesors>2</Procesors>
      <MaxMemory>1,433,403,392</MaxMemory>
      <FreeMemory>21,288,896</FreeMemory>
      <TotalMemory>40,370,176</TotalMemory>
   </GetJvmInfo>
   <GetUseInfo Status="200" Reason="OK">
      <SffNioServer>Total(connections/reads/writes): (1/0/0);
Time(total/per
        read): (116/0.000) ms; read: 0 bytes; written: 0
bytes</SffNioServer>
      <NioServerSocketChannel>2</NioServerSocketChannel>
      <NioAcceptedSocketChannel>1</NioAcceptedSocketChannel>
```

5.2.3 How to deploy a FacetScript to TQLEngine

In order to deploy the FacetScript to TQLEngine, you can simply post the content of the FacetScript using the fid-SffFacetAgentFacet endpoint.



```
<GetInfo/>
```

```
<FacetAgent Status="200" Reason="OK" Version="1.0.0.201503260009">
    <GetJvmInfo Status="200" Reason="OK">
        <Procesors>2</Procesors>
        <MaxMemory>1,433,403,392</MaxMemory>
        <FreeMemory>21,288,896</FreeMemory>
        <TotalMemory>40,370,176</TotalMemory>
    </GetJvmInfo>
    <GetUseInfo Status="200" Reason="OK">
        <SffNioServer>Total(connections/reads/writes): (1/0/0); Time(total/per read): (116/0.000) ms; read: 0 bytes; written: 0
bytes</SffNioServer>
        <NioServerSocketChannel>2</NioServerSocketChannel>
        <NioAcceptedSocketChannel>1</NioAcceptedSocketChannel>
        <TotalChannels>3</TotalChannels>
    </GetUseInfo>
    <GetFacetInfo Status="200" Reason="OK">
        <SffFacetAgentFacet ActiveSince="1427745290392" ActiveSinceDate="Mon Mar 30 15:54:50 EDT 2015"/>
```

A. Practical Examples

In this section we provide various partical examples.

A.1 How to enable access to static files in your local filesystem over an HTTP Transport

Deploy a new Facetinstance with SffHttpStaticFileFacet to provide access to your local filesystem over an HTTP endpoint.

```
<NewFacetInstance fid="file" name="File" type="SffHttpStaticFileFacet">
        <OnActivate>
            <Process BasePath="web" Resource="res"/>
        </OnActivate>
    </NewFacetInstance>
```

A.2 How to create a simple ThingFacet Script to enable Thing Queries (TQL) over an HTTP Transport

The simplest way to enable TQL Queries over an HTTP transport is to deploy a new facet instance with SffTqlFacet type. In the OnActivation flow you can specify the reference to the data/thing model using <Include> tag.

Example:

```
<NewPackage>
   <!--
===========================================================================
==================== -->
   <RESTHeaders>
      <Headers>
         <Connection>close</Connection>
         <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>
         <Access-Control-Allow-Methods>GET, POST, PUT
         </Access-Control-Allow-Methods>
      </Headers>
   </RESTHeaders>

   <GenerateMessage>
      <Message type="xml">
         <Value>
            <Include>GenMsgArgs</Include>
         </Value>
      </Message>
   </GenerateMessage>

   <NewFacetInstance fid="fleet" name="TQL" type="SffTqlFacet">
      <OnActivate>
         <Process>
            <Storage name="TQLStudio" type="SqlSff"
               comment="TQLStudio SFF Unstructured SQL database" />
            <Namespace>

   <Include>/atomiton/tqlengine/fleetmanagement/dbm/fleet.dbm.xml
               </Include>
            </Namespace>
         </Process>
      </OnActivate>

      <OnOpen ModifyPipeline="HttpServerExtensionArgs" />

      <OnRequest>
         <DoRequest>
            <Process Return="CMD_NOP">
               <Message type="xml">
                  <Value>
                     <Include>$Request.Message.Value</Include>
                  </Value>
               </Message>
            </Process>
         </DoRequest>

         <DoResponse>
            <Process>
               <Include>RESTHeaders</Include>
               <GenMsgArgs>
                  <Include>$Response.Message.Value</Include>
               </GenMsgArgs>
               <Include>GenerateMessage</Include>
            </Process>
         </DoResponse>
      </OnRequest>
   </NewFacetInstance>
```

```
</NewPackage>
```

A.3 How to start a WebSocket Server

You can enable Websocket server at a given port by deploying another FacetInstance.

Example: The websocket server will be started on port 9090 at URL ws://localhost:9090/fid-ws

```
<NewFacetInstance fid="ws" name="WebSocket" type="SffTcpFacet">
      <OnActivate>
         <Process Server.Port="9090" />
      </OnActivate>
      <OnOpen ModifyPipeline="WebSocketServerExtensionArgs" context="keep"
/>
      <OnRequest>
         <DoResponse>
            <Process>
               <Message type="text">
                  <Value>[:$Request.Message.Value:]</Value>
               </Message>
            </Process>
         </DoResponse>
      </OnRequest>
   </NewFacetInstance>
```

A.4 An example : Smart Parking

A.4.1 SmartParking Data Model File (ParkingLotAssetsTQL.dbm.xml)

```
<!--
========================================================================
   ** The purpose of this file is to maintain all the data models
associated with
  ** smart parking. **
========================================================================
-->
<Namespace name="ParkingLot">
   <DataModel name="ParkingLotTQLAssets">
      <DataFacet name="Organization">
         <String name="id" />
         <String name="Industry" />
         <String name="Name1" />
```

```
        <String name="LegalName" />
        <String name="URL" /><!-- fully qualified web address -->
        <String name="AddressCountry" />
        <String name="AddressState" />
        <String name="AddressPostalCode" />
        <String name="AddressStreetName1" />
        <String name="AddressStreetName2" />
        <String name="AddressStreetNumber" />
        <String name="ContactPerson" />
        <String name="Department" /><!-- Sales -->
        <String name="Email" /><!-- info@mycomany.com -->
        <String name="PhoneNumber" /><!-- (408) 444-7777 -->
        <String name="GlobalLocationNumber" /><!-- 348485959 -->
        <String name="Logo" /><!-- url to image for logo -->
        <Double name="LocAlt" /><!-- Altitude -->
        <Double name="LocLon" /><!-- Longitude -->
        <Double name="LocLat" /><!-- Altitude -->
        <!-- Unique name="Organization" value="id" /> -->
     </DataFacet>

     <DataFacet name="GeoCordinate">
        <String name="id"/>
        <Double name="minLat"/>
        <Double name="maxLat"/>
        <Double name="minLon"/>
        <Double name="maxLon"/>
     </DataFacet>

     <DataFacet name="DefParkingSpaceState">
        <String name="id"/>
        <Integer name="vacant"/>
        <Integer name="occupied"/>
        <Integer name="reserved"/>
        <Integer name="total"/>
        <Integer name="available"/>
     </DataFacet>

     <DataFacet name="DefParkingPolicyType">
        <String name="id"/>
        <Integer name="loadingZone"/>
        <Integer name="noParking"/>
        <Integer name="alwaysValid"/>
        <Integer name="fireZone"/>
        <Integer name="standingZone"/>
```

```
</DataFacet>

<DataFacet name="DefParkingSpotSize">
   <String name="id"/>
   <Integer name="compact"/>
   <Integer name="regular"/>
   <Integer name="large"/>
   <Integer name="handicap"/>
</DataFacet>

<DataFacet name="DefParkingSpotOrientation">
   <String name="id"/>
   <Integer name="single"/>
   <Integer name="multi"/>
   <Integer name="perpendicular"/>
   <Integer name="parallel"/>
   <Integer name="angle"/>
</DataFacet>

<DataFacet name="DefParkingUsageType">
   <String name="id"/>
   <Integer name="self"/>
   <Integer name="parknRide"/>
   <Integer name="valet"/>
   <Integer name="airport"/>
   <Integer name="meetnGreet"/>
   <Integer name="parknFly"/>
</DataFacet>

<DataFacet name="DefParkingMeteringType">
   <String name="id"/>
   <Integer name="singleMeter"/>
   <Integer name="multiMeter"/>
   <Integer name="timebased"/>
   <Integer name="nonMetered"/>
</DataFacet>

<DataFacet name="DefParkingRate">
   <String name="id"/>
   <Integer name="durationMinutes"/>
   <Integer name="farePerMinute"/>
   <Integer name="minimumFare"/>
   <Integer name="maximumFare"/>
   <Integer name="currentFare"/>
```

```
      </DataFacet>

      <DataFacet name="ParkingFloorState">
         <String name="id"/>
         <String name="vacant"/>
         <String name="occupied"/>
         <String name="total"/>
         <String name="available"/>
         <String name="reserved"/>
      </DataFacet>

      <!-- Main Table that is going to hold the spots -->
      <DataFacet name="ParkingSpot">
         <String name="id"/>
         <String name="parkingFloorId"/>
         <String name="geoCordinateId"/>
         <String name="label"/>
         <String name="defParkingSpotSizeId"/> <!-- Emulate Foreign key -->
         <String name="state"/> <!-- Emulate Foreign key -->
      </DataFacet>

      <DataFacet name="ParkingFloor">
         <String name="id"/>
         <String name="ParkingFloorStateId"/>
         <String name="defParkingUsageTypeId"/>
         <String name="floorNumber"/>
         <String name="digitalSignLabel1"/>
         <String name="digitalSignLabel2"/>
         <String name="defParkingPolicyTypeId"/>
         <String name="defParkingRateId"/>
         <String name="operatedByOrgId"/> <!-- References Organization.id
Table -->
      </DataFacet>

      <!-- ============Metering Related======== -->
      <DataFacet name="ParkingMeter">
         <String name="id"/>
         <String name="parkingFloorId"/>
         <String name="parkingSpotId"/>
         <String name="geoCordinateId"/>
         <String name="parkingMeterLabel"/>
      </DataFacet>
```

```
<!-- ==========Light Related ============== -->
<DataFacet name="GeoSite">
   <String name="id"/>
   <String name="zone"/>
   <String name="route"/>
   <String name="street"/>
   <String name="block"/>
   <String name="parkingFloorId"/>
   <String name="parkingSpotId"/> <!-- Added to facilitate demo -->
</DataFacet>

<DataFacet name="AreaLight">
   <String name="id"/>
   <String name="powerState"/>
   <Double name="intensityLevel"/>
   <String name="operatedByOrg"/>
   <String name="block"/>
   <String name="parkingFloorId"/>
   <String name="parkingSpotId"/> <!-- Added to facilitate demo -->
</DataFacet>

<DataFacet name="stallLight">
   <String name="id"/>
   <String name="powerState"/>
   <Double name="intensityLevel"/>
   <String name="operatedByOrg"/>
   <String name="block"/>
   <String name="parkingFloorId"/>
   <String name="parkingSpotId"/> <!-- Added to facilitate demo -->
</DataFacet>

<DataFacet name="camera">
   <String name="id"/>
   <String name="url"/>
   <String name="state"/>
   <String name="parkingFloorId"/>
   <String name="parkingSpotId"/>
</DataFacet>

<DataFacet name="magneticSensor">
   <String name="id"/>
   <String name="operatedByOrgId"/>
   <String name="state"/>
   <String name="parkingSpotId"/>
```

```
      </DataFacet>

      <DataFacet name="scorecard">
         <String name="id"/>
         <String name="scard"/>
      </DataFacet>
   </DataModel>
</Namespace>
```

A.4.2 FacetScript source code.

Manage ParkingLot TQLs. (TQLInterface.mqp.xml)

```
<!-- The purpose of this FacetInstance is to install the SmartPakring DB
   interface -->

<NewPackage>
      <RESTHeaders>
      <Headers>
         <Connection>close</Connection>
         <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>
         <Access-Control-Allow-Methods>GET, POST, PUT
         </Access-Control-Allow-Methods>
         <Access-Control-Allow-Headers>Content-Type
         </Access-Control-Allow-Headers>
      </Headers>
   </RESTHeaders>

   <INPUTHeaders>
      <Headers>
         <Content-Type>application/xml</Content-Type>
      </Headers>
   </INPUTHeaders>
   <!--
==============================================================================
==================== -->
   <GenerateMessage>
      <Message type="xml">
         <Value>
            <Include>GenMsgArgs</Include>
         </Value>
      </Message>
   </GenerateMessage>
   <SetAreaLightMessage>
      <!-- AreaLight -->
      <set name="UpdateAreaLight"
target="[:[:alParam:].parkingSpotId.Value:]"
         time="[:$Now(%d):]" value="[:[:alParam:].intensityLevel.Value:]" />
   </SetAreaLightMessage>

   <SetStallLightMessage>
      <!-- StallLight -->
      <set name="UpdateStallLight"
         target="[:[:alParam:].parkingSpotId.Value:]"
         time="[:$Now(%d):]" value="[:[:alParam:].powerState.Value:]" />
   </SetStallLightMessage>

   <SetParkingMeterMessage>
      <!-- ParkingMeter -->
      <set name="UpdateParkingMeter"
         target="[:[:alParam:].parkingSpotId.Value:]"
         time="[:$Now(%d):]"
         value="[:[:alParam:].parkingMeterLabel.Value:]" />
   </SetParkingMeterMessage>

   <SetDigitalSignMessage1>
      <!-- DigitalSignage -->
      <set name="UpdateParkingFloor"

   target="[:$Response.Message.Value.Update.ParkingFloor.id.Value:].digital
SignLabel1"
         time="[:$Now(%d):]"

   value="[:$Response.Message.Value.Update.ParkingFloor.digitalSignLabel1.V
alue:]" />
   </SetDigitalSignMessage1>

   <SetDigitalSignMessage2>
      <!-- DigitalSignage -->
      <set name="UpdateParkingFloor"

   target="[:$Response.Message.Value.Update.ParkingFloor.id.Value:].digital
SignLabel2"
         time="[:$Now(%d):]"
   value="[:$Response.Message.Value.Update.ParkingFloor.digitalSignLabel2.V
alue:]" />
   </SetDigitalSignMessage2>

   <!--
===========================================================================
==================== -->


   <NewFacetInstance fid="tqlinterface" name="TQL"
      type="SffTqlFacet">
      <OnActivate>
         <Process>
            <Storage name="TqlTest" type="SqlSff" comment="SFF Unstructured
SQL database" />
            <Namespace>

   <Include>/atomiton/tqlengine/parkinglot/dbm/ParkingLotAssetsTQL.dbm.xml
               </Include>
            </Namespace>
         </Process>
      </OnActivate>

      <OnOpen ModifyPipeline="HttpServerExtensionArgs" />

      <OnRequest>
         <Log>Request is: [:$Request.Message.Value:]</Log>
         <DoRequest>
            <Process Return="CMD_NOP">
               <Message type="xml">
                  <Value>
                     <Include>$Request.Message.Value</Include>
                  </Value>
               </Message>
            </Process>
         </DoRequest>
         <DoResponse>
            <if condition="$Request.Message.Value/boolean(Query/Update)">
               <then>
                  <if
condition="$Response.Message.Value/count(Update/AreaLight) gt 0">
                     <then>
                        <!-- For ... -->
                        <for each="al"
in="$Response.Message.Value.Update.AreaLight">
                           <Log>Looping....</Log>
                           <DoResponse target="ws:*" disable="CMD_SERVER">
                              <Process>
                                 <Now>[:[:@Request:]$Now(%d):]</Now>
                                 <alParam>$LocalData.al</alParam>
                                 <Message type="text">
                                    <Value>
                                       <Include>SetAreaLightMessage</Include>
                                    </Value>
                                 </Message>
                              </Process>
                           </DoResponse>
                        </for>
                     </then>
                  </if>
                  <if
condition="$Response.Message.Value/count(Update/StallLight) gt 0">
                     <then>
                        <for each="sl"
in="$Response.Message.Value.Update.StallLight">
                           <DoResponse target="ws:*" disable="CMD_SERVER">
                              <Process>
                                 <Now>[:[:@Request:]$Now(%d):]</Now>
                                 <alParam>$LocalData.sl</alParam>
                                 <Message type="text">
                                    <Value>
                                       <Include>SetStallLightMessage</Include>
                                    </Value>
                                 </Message>
                              </Process>
                           </DoResponse>
                        </for>
                     </then>
                  </if>
                  <if
condition="$Response.Message.Value/count(Update/ParkingMeter) gt 0">
                     <then>
                        <for each="pm"
in="$Response.Message.Value.Update.ParkingMeter">
                           <DoResponse target="ws:*" disable="CMD_SERVER">
                              <Process>
                                 <Now>[:[:@Request:]$Now(%d):]</Now>
                                 <alParam>$LocalData.pm</alParam>
                                 <Message type="text">
                                    <Value>

      <Include>SetParkingMeterMessage</Include>
                                    </Value>
                                 </Message>
                              </Process>
                           </DoResponse>
                        </for>
                     </then>
                  </if>
                  <if
condition="$Response.Message.Value/boolean(Update/ParkingFloor)">
                     <then>
                        <DoResponse target="ws:*" disable="CMD_SERVER">
                           <Process>
                              <Now>[:[:@Request:]$Now(%d):]</Now>
                              <Message type="text">
                                 <Value>
                                    <Include>SetDigitalSignMessage1</Include>
                                 </Value>
                              </Message>
                           </Process>
                        </DoResponse>
                     </then>
                  </if>
                  <if
condition="$Response.Message.Value/boolean(Update/ParkingFloor)">
                     <then>
                        <DoResponse target="ws:*" disable="CMD_SERVER">
                           <Process>
                              <Now>[:[:@Request:]$Now(%d):]</Now>
                              <Message type="text">
                                 <Value>
                                    <Include>SetDigitalSignMessage2</Include>
                                 </Value>
                              </Message>
                           </Process>
                        </DoResponse>
                     </then>
                  </if>
               </then>
            </if>
```

```
            <Process>
              <Include>RESTHeaders</Include>
              <GenMsgArgs>
                <Include>$Response.Message.Value</Include>
              </GenMsgArgs>
              <Include>GenerateMessage</Include>
            </Process>
          </DoResponse>

        </OnRequest>
      </NewFacetInstance>

    </NewPackage>
```