2015

Global Azure

BOOTCAMP

Simple IoT with Azure Mobile Services

David Jones
Sportronics

e101
embedded101

David Jones

Embedded MVP
Melbourne, Victoria, Australia
@CEDriverWiz
davidjones@sportronics.com.au

e101
embedded101

Agenda

- Microsoft Azure Mobile Service
- AzMS HTML POST & GET
- AzMS Telemetry Universal app
- AzMS version 1 Tables
- JSON
- CEJSON, Ardjson
- Post, Get, Update, Delete of telemetry data with AzMS
- CEJSON Parsing of HTML Response from AzMS
- AzMS Server side scripting

e101
embedded101

Simple Scenario

The Cloud

Users

Sensors

Actuators

e101
embedded101

Simple Take Homes

- Internet of **Simple** Things is **simple** with Azure Mobile Services (AzMS).
- AzMS uses standard HTML posts and queries
- AzMS communications don't need SSLs which is a Cloud enabler for many embedded devices.
- When you create an AzMS service you get an option to create a sample Universal app that can communicate with an AzMS table (ToDo app). This app can run on Windows, Surface, Windows Phone, Android and IOS.
  - A variant of this to submit, read and modify telemetry data was covered.
- The command line tool cURL is useful for testing the AzMS query syntax and API.
  - Some examples were given.
- CEJSON and Ardjson, as available with full source, are libraries that **simplify** things further by abstracting the AzMS communication with **simple** devices (Windows Embedded Compact/CE and Arduino).
  - A sensor collecting temperature and pressure data, submitting it to AzMS and then reading and modify that data with and to an Arduino device was demonstrated using Ardjson
  - Similarly, data was communicated with AzMS using CEJSON from an 86Duion device, as well as from a Windows console app..
- Data received from AzMS is in JSON format which is **simple** to parse and is implemented as stream parser in CEJSON and Ardjson.
  - This was demonstrated on 86Duino and Arduino devices using the response from GET queries.
- Some AzMS hints covered:
  - Create tables with JavaScript backend not .NET to keep things simpler
  - Use version 1 AzMS tables instead of the default version 1 to minimize the data received. (Use integer id field instead of a GUID).
    - This can't be done in the Azure Portal so covered how to do this.
  - AzMS server side scripts intercept queries posted and can also modify the response. A custom set of Select queries is **simplest** to implement in AzMS GET queries.
    - Some example scripts for GET and POST were provided.

e101
embedded101

Microsoft Azure Mobile Service

- **Simple** cloud backend to mobile apps
  - Backend can be JavaScript or .NET
- 24 x7 availability
- Store data in SQL, Table Storage, and MongoDB
- “Edge” can be anything that produces/consumes JSON over HTML
  - Windows Desktop, tablet, phone, handheld, embedded
  - IOS, Android
  - HTML
- Can use cloud-based sync to build apps that work offline
- Push notifications
- Simple sign-on

e101
embedded101

AzMS Tables

- Use Azure MS SQL Database for backend
- Create an AzMS table and a mirror MS SQL table is created.
- AzMS table is more “free-form” than MS SQL
  - Don’t need to define table fields
  - As records are posted, new fields are created to match as required.
    - For deployment this can be locked down.
  - Can simply add auto-generated fields in POST script
  - Can script field validation in POST as well
- Id field: Primary key
  - Version 2 tables (default) 32 character GUID string
    - Auto-generate creation and modified dates
  - Version 1 tables (need to create manually) use auto-BigInt

e101 embedded101

AzMS REST Queries

- Data is posted, updated, retrieved and deleted using HTML messages to the service.
- Posted data is a single JSON object string
- Retrieved data is returned as a JSON array of objects string
  - No nesting of object s (eg arrays within arrays). KIS

AzMS HTML Queries

- POST: New record
- GET: Get table contents
  - Can apply filters to enrich the selection
- POST: Use PATCH
  - Requires the record id and modified name-values
  - Can only update one record, can’t update a selection
- DELETE
  - Requires the record id
  - Can only delete one record, can’t delete a selection

AzMS Query Parameters

- A query requires:
  - From the Azure Portal:
    - The AppKey
    - The AzMS URL
    - The table name (needs to exist)
  - The HTML Verb
  - Any data specific to the verb
    - In JSON format
  - The content type:
    - Set to json

An AzMS POST Query

```
POST /tables/telemetry2 HTTP/1.1
Host: sportronicsdj.azure-mobile.net
X-ZUMO-APPLICATION:
NtcMLvQtuAqWtvXOwwZVQtpHevNUnN97
Content-Type: application/json
{"sensor":"temperature","value":27}
Content-Length: 31
```

A GET Response

```
[{id:56,
"sensor":"Temperature","value":56},{id:63,
"sensor":"Humidity",
"value":90},{id:78,"sensor":"Pressure",
"value":1002}]
```

- The Response is a one line string

cURL.exe

- curl is a command line tool and library
- For transferring data with URL syntax
- Can send HTML post and queries from a command line.
- Windows version download available.
  - See links at end.
- Useful for establishing correct syntax for messages with a service
- cURL can send POST GET PUT (PATCH) DELETE and apply filters
  - See “cURL CRUD Examples” in links

e101
embedded101

cURL: POST a Name-Value pair to AzMS

```
curl -v -X POST -H Content-Type:application/json
-H X-ZUMO-APPLICATION:
 NtcMLvQtuAqWtvXOwwZVQtpHevNUnN97
-d "{ \"sensor\":\"Temp1\",\"value\" : 42 }"
```

http://sportronicsdj.azure-mobile.net/tables/telemetry2

Note: All on one line

e101 embedded101

cURL: GET Query

```
curl -v -X GET -H Content-Type:application/json
-H X-ZUMO-APPLICATION:%AppKey%
"%AzureMobileServiceURL%/tables/%MSTable%
```

- Note: All on one line
- This is inside a batch file with Appkey etc declared earlier in the script.
- Can apply filters to the query
- Eg Append
  ?$filter=(startswith(sensor,'Temperature'))"

e101
embedded101

Demo: cURL

e101
embedded101

Azms Telemetry Universal App

- When you create an AzMS service you get an option to create a sample ToDo Universal App that posts ToDo items to an AzMS table.
  - These are updated by checking tehethe items in the app.
  - Only incomplete (unchecked) items are displayed.
  - Available for a variety of platforms
- The CEJSON/Ardjson Codeplex projects have a Telemetry app that has been morphed from the ToDo app:
  - Post telemetry data (sensor name and sensor value)
  - Retrieve telemetry data not marked as complete
  - Scan sensor data updating the table so that only the latest sensor values are tagged as incomplete.
    - Embedded devices then only get the relevant records.. Smaller.

e101
embedded101

MICROSOFT AZURE MOBILE SERVICES

Sensor Mobile Service

1

Insert a telemetry data

Enter a sensor and its value and click Save to insert a new entry into the Telemetry database

Sensor: UV Value: 70 Save

2

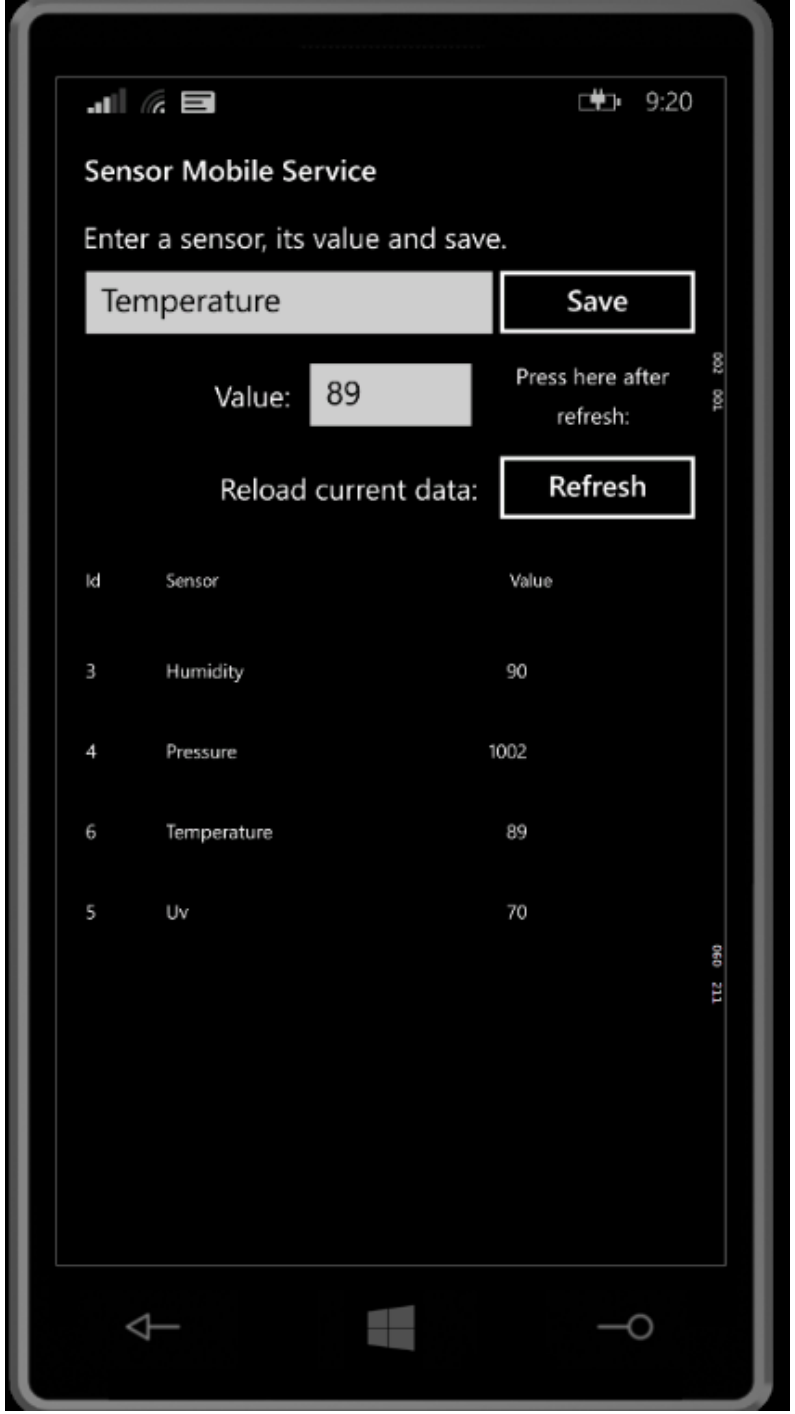
Query and Update Data

Click refresh below to load the current active data from the Telemeetry database. Use the checkbox to tag a datum as stale and update the database

Refresh

Id	Sensor	Value
3	Humidity	90
4	Pressure	1002
2	Temperature	67
5	Uv	70

Telemetry App on Phone



e101 embedded101

Demo: Telemetry App + AzMS Tables

CEJSON: https://cejson.codeplex.com/

Ardjson: https://ardjson.codeplex.com/

e101
embedded101

The story so far ..

We can simply interact with an AzMS Table using .NET and Universal Apps

But what about embedded devices such as Windows Embedded Compact/CE and Arduino?

e101
embedded101

How to send these queries and get the response from embedded devices.

How to interpret the response?

Embedded Issues

- Don’t have a REST API
- Resource restrictions
  - Can’t blindly download the whole of a table
    - Memory overload
- Typically only want to
  - Read a sensor value from hardware
  - Upload a single name-value pair
  - Download only latest values for sensors
  - Actuate an actuator (eg Relay) if a sensor is above an alarm value

e101
embedded101

AzMS Version 1 Tables

- By default AzMS Portal currently creates Version 2 tables auto-generating a 32 byte GUID string as the id (and primary key) field.
- This is overkill for resource starved devices (such as Arduino).
- Can use Version 1 tables instead but need to manually create form outside of the Azure Portal
  - Use an auto-generated (Big)Int
- Much better for GET

e101
embedded101

Version1 Table: Example

- Requires Microsoft Cross Platform Command Line Tools (See Links)

```
Prompt>azure account download
info:    Executing command account download
info:    Launching browser to http://go.microsoft.com/fwlink/?LinkId=254432
help:    Save the downloaded file, then execute the command
help:      account import <file>
info:    account download command OK

Prompt>azure account import  c:\azure\credentials.publishsetting
info:    Executing command account import
info:    account import command OK

Prompt>azure mobile table create --integerId sportronicsdj telemetry2
info:    Executing command mobile table create
+ Creating table
info:    mobile table create command OK

Prompt>
```

Login with Azure credentials

Save the file when prompted without spaces

Service name is name of service not URL

e101 embedded101

A Version 1 Table

telemetry2

Browse Script Columns Permissions

id	sensor	value
1	Temperature1	562
2	Temperature2	134
3	Humidty1	67
4	Humidty2	78
5	Temperature1	926

e101
embedded101

Demo: AzMS Version 1 Tables

About Version 1 AzMS Tables: http://tinyurl.com/kkq94xv
MS Cross-platform Command Line Tools Installer: http://tinyurl.com/pobnx8j

e101
embedded101

REST

- HTML queries with AzMS are REST
  - Representational State Transfer.
  - It relies on a stateless, client-server, cacheable communications protocol
  - REST is a simple alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL).

- Uses JSON to format data.

e101
embedded101

JSON

- **J**ava**S**cript **O**bject **N**otation
- A lightweight data-interchange format.
- Simple for humans to read and write.
- Simple for machines to parse and generate
  - Many libraries available
- Lighter than XML
- Simple to pass over HTTP being textual
- Has two entities:
  - Collection of name value pairs (object, record etc.)
  - Ordered list (array etc.)

e101
embedded101

JSON Entities

- JSON Object (Record)
  - Colon separated name value pair
  - {Name:Value}
- JSON Array
  - Comma separated list of values
  - [Value,Value,Value…]
- JSON Array of Objects
  - [{},{],{},…]
- Name: A string
- Value: string,bool,int,float .. *No DateTime*

JSON Examples

Object

```
{"Name":"Jim", "Age":27, "Address":"Somewhere"}
```

```
{"date":"12/01/2015","sensor":"temperature1","alarm"
:"hi","resolvedq":"true"}
```

Array

```
[1,2,3,4]
```

```
["John", "Sue",Harry]
```

```
[{"Temp",23.5},{"Temp",23.6},{"Temp",33.2}]
```

e101
embedded101

Embedded REST

- Need to be able to send restful requests from device over HTTP: GET, POST, PATCH and DELETE
- Need to get the response:
  - Check for success or failure
  - When data returned (as JSON) need to parse it into records.
- Arduino
  - Ethernet library supports HTTP REST messaging
- Windows Embedded Compact/CE
  - Can implement HTTP requests using sockets

Ardjson

- https://Ardjson.Codeplex.com
- A project for submitting (POST) telemetry data to an AzMS table
  - Version 1 tables
- Can also GET, PATCH and DELETE
- Implements JSON stream parser for interpreting HTTP Response.
- Can read data from Arduino sensors
- Memory was a big issue with this device so had to optimise substantially.

e101
embedded101

Demo: Ardjson

https://Ardjson.Codeplex.com

e101
embedded101

CEJSON

- https://CEJSON.Codeplex.com
- A project for submitting (POST) telemetry data to an AzMS table
  - Version 1 tables
- Can also GET, PATCH and DELETE
- Implements JSON stream parser for interpreting HTTP Response.
- Available as an app with command line options
- Also as a DLL library (under development). ..Extensible
- Windows Console Version as well Compact 2013
  - Simple to port to Compact 7 and CE6

e101
embedded101

CEJSON Command Line

CEJSONApp {GET|POST|PATCH|DELETE} {Filter} {id} {Field}/Sensor name} {Sensor/Field value}
CEJSONApp GET
CEJSONApp GET <Filter>
Filter can be any valid AzMS Filter. Some telemetry2 specific ones:
CEJSONApp GET embedded={1|2|3}
1 is the default
CEJSONApp GET sensor=<Sensor name>

CEJSONApp POST <Sensor name> <int Sensor value>
CEJSONApp {PATCH|PUT} <id> <Field name> <int Field value>
CEJSONApp DELETE <id>

- Only first 3 characters of verbs are important.
- ie Can be GET POS PAT PUT or DEL

- Defaults to GET.

e101
embedded101

Parsing of JSON Response

- JSON is human readable so creating a parser is not to difficult.
- Initially with Ardjson the whole response was save then parsed.
- Was simple to run out of RAM
- So stream parser as a state machine was developed.
- Parser implemented as a function that takes one character
- The state of the parse is maintained between calls.
- Its essentially one big switch statement based upon the current state

JSON Parser States

startOfArray,
startOfRecord,
startOfName,
gettingName,
nameValueSeparator,
startOfValue,
gettingValue,
gettingEndOfValueORRecord,
gotEndOfRecord,
gettingRecordSeparator,

done,
error,
gettingString,
gettingBoolean,
gettingInteger,
gettingFloat,
gettingNull

Demo: CEJSON
Desktop and 86Duino (Compact 2013)

https://CEJSON.Codeplex.com

e101
embedded101

CEJSON Documentation as Blogged on Embedded101.com

- CEJSON - 1: IoT on Windows Embedded Compact with Azure Mobile Services
- CEJSON - 2: IoT on WEC 2013 with AzMS – CRUD Primitives
- CEJSON - 3: IoT and Azure Mobile Service Scripts
- CEJSON – 4: IoT Version 1 Azure Mobile Service Tables
- CEJSON – 5: IoT The JSON Parser
- CEJSON – 5.5: The Parser source code in detail
- Index to above: http://cejson.codeplex.com/documentation

e101
embedded101

AzMS Server Side Scripting

- AzMS supports filters as part of HTTP GET
  - See “cURL CRUD Examples” (See links)
- Can be simpler to implement a set of “allowed” queries on the server side.
- AzMS has a script for each of POST, GET, PATCH and DELETE which can be modified
  - Insert, Read, Update and Delete respectively
  - Can modify via Azure Portal
    - No cur and paste
  - Simpler to access from Visual Studio

e101
embedded101

AzMS Script Uses

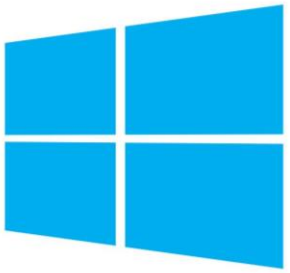
- POST
  - Add additional fields or auto-set field values
    - Create DateTime
    - Record as fresh (complete=false)
    - ‘Normalise’ sensor names
    - Validate valuse
- GET
  - Restrict records to only those incomplete
  - Vary the fields returned
  - Return all records for a specific sensor

e101
embedded101

Demo: telemetry2 Scripts

Show Read and Post scripts

e101
embedded101

The Future:  Windows 10

Desktop, Tablet, Industry	Universal Apps
Handheld & Phone	
Embedded / IOT	

e101
embedded101

One Windows Platform

Links

- Azure: http://azure.microsoft.com/en-us/
- CEJSON: https://cejson.codeplex.com/
- Ardjson: https://ardjson.codeplex.com/
- cURL: http://curl.haxx.se/
- “cURL CRUD Examples”: http://tinyurl.com/nzxb3sr
- About Version 1 AzMS Tables: http://tinyurl.com/kkq94xv
- MS Cross-platform Command Line Tools Installer: http://tinyurl.com/pobnx8j
- http://cejson.codeplex.com/documentation

e101
embedded101

David Jones

Embedded MVP
Melbourne, Victoria, Australia
@CEDriverWiz
davidjones@sportronics.com.au

e101
embedded101

e101
embedded101