AUTODESK **UNIVERSITY** 2015

SD11048

# Connecting Desktop and Cloud

Jeremy Tammik
Autodesk Inc.

## Learning Objectives

- Connect desktop application with cloud and mobile apps
- Architect a completely portable cloud-based data repository, geometry viewer and navigator
- Demonstrate use of node.js, WebGL and the Autodesk View and Data API
- Implement a basic generic universal component usage analysis, visualisation and reporting app

## Description

We describe the nitty-gritty programming and implementation details of a cloud-based system to analyse, visualise and report on universal component usage. The components can be Revit families used in BIM or any other kind of assets in any other kind of system. The focus is on the cloud-based database used to manage the component occurrences. Models are visualised using pure WebGL, Three.js and the Autodesk View and Data API, providing support for online viewing and model navigation. The web app can be used in any browser and on any mobile device with no need to install any additional software whatsoever. This is an advanced class for experienced programmers.

## Your AU Expert

***Jeremy Tammik*** *is The 3D Web Coder and The Building Coder. He works with the Autodesk Developer Network (ADN) team on Autodesk APIs and web services, providing developer support, training, conferences, presentations, and blogging on the Revit API and cloud and mobile technologies.*

*He joined Autodesk in 1988 as a technology evangelist driving AutoCAD application development in Europe, USA, Australia, and Africa. He co-founded ADGE, the AutoCAD Developer Group Europe, and is a prolific author. He left Autodesk in 1994 to work as an independent HVAC (heating, ventilating, and air conditioning) application developer and re-joined the company in 2005.*

*Jeremy graduated with degrees in mathematics and physics from the Philipp University of Marburg in Germany. He worked as a teacher and translator of both computer and human languages and as a C++ programmer on early GUI and multitasking projects.*

*Jeremy is fluent in six European languages, vegetarian, has four kids, one grandchild, loves cooking, climbing, hiking, sports, nature, music, theatre, literature, art, photography, travel, mountains, oceans, meditation, spirituality, yoga, carpentry, adventure, survival, problem solving and challenges of all kinds.*

## Table of Contents



## Introduction

Let's discuss connecting the desktop with the cloud.

As a sample software platform, I will be looking at Revit on the desktop, web servers and NoSQL databases in the cloud.

## What Cloud are we Talking About?

Please note that you yourself define exactly what and where the cloud is located and who can access it.

Everything I demonstrate here can be installed locally on your own machine – no strings attached – only you can access it – or on an intranet, extranet, or published to the entire world.

Obviously, the more widely accessible it is, the more widely useful it can potentially end up being.

In case you publish it openly, you can restrict access rights as you wish and get involved with security issues. I will not deal with that aspect of things, though.

## Message and Takeaway

The main point I aim to prove to you during this session is:

*It is easy to hook up a Revit or any other desktop application with the cloud.*

The possibilities are powerful and infinite.

All the web tools and functionality you can ever possibly need are available already, online, for free, provided by an abundance of open source technology stacks, projects and libraries.

In this presentation, I demonstrate and discuss a several simple working examples: my 2D room editor, FireRating in the Cloud, my current work in progress, CompHound, and a View and Data API sample.

You can grab these samples to get started implementing your own ideas really fast.

My underlying research is extensively documented, so it should help you circumvent the numerous

snags I hit and resolved.

To reiterate:

- This is really easy.
- This can be really powerful.

## Four Samples

As said, I will present the following four samples, and they will each lead to further topics:

- The 2D cloud-based round-trip room editor
- FireRating in the cloud
- The CompHound component tracker
- Dotty animated 3D assembly instructions

Let's dive right in:

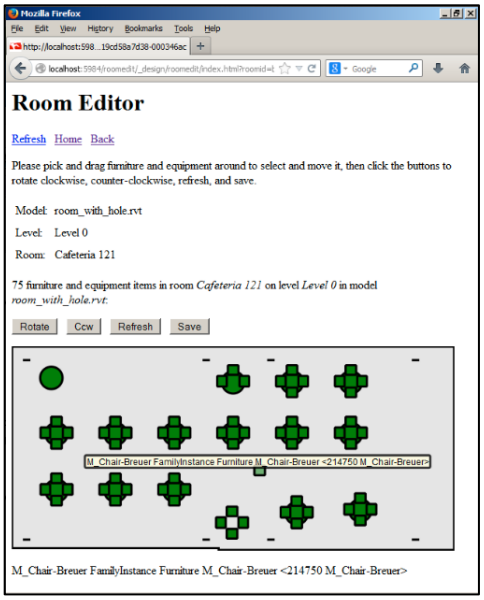
## The 2D Cloud-Based Round-Trip Room Editor

This project was my first serious exploration of a desktop-cloud connection.

I initially worked on it in 2013, presented it at two Autodesk Tech Summit conferences and at Autodesk University 2013, in the session DV1736 – *Cloud-Based, Real-Time, Round-Trip, 2D Revit Model Editing on Any Mobile Device*.

It consists of two components:

- The RoomEditorApp Revit add-in
- The roomedit CouchDB web-based NoSQL database




It demonstrates bi-directional data exchange between the two, i.e., between a Revit BIM and a globally accessible cloud-based web database, usable on any device, in any browser.

The display is implemented based on SVG, scalable vector graphics. The interface supports position editing by 2D dragging.

The Revit add-in captures a simplified plan view of rooms and the furniture contained in them and exports that to the web database.

It captures the containment and relationship hierarchy including Project → Level → Room → Furniture = family instance → family symbol.

For the latter three, it also captures the 2D geometry:

- Room: 2D boundary loops.
- Furniture instance: 2D transform, i.e. location and rotation.

- Furniture family symbol: XY-projected 2D boundary loops.

The geometry is encoded in SVG strings, enabling easy visualisation in any browser and on any device.

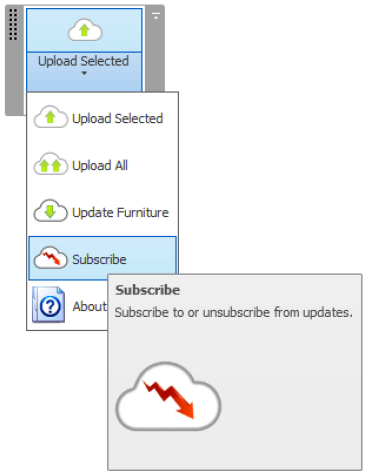
This data is stored in the database and displayed in the browser, supporting the following steps:

- Navigate through the Project → Level → Room containment hierarchy
- Display a 2D view of a selected room with the furniture it contains
- Click (or touch) and drag to modify the furniture rotation and location within the room
- Edit any of the writeable furniture Revit parameters

The main point is still to come, though:

Besides simply exporting the BIM data in the UPLOAD ROOMS and UPLOAD ALL ROOMS external commands, the Revit add-in implements two more:

- UPDATE FURNITURE: reimport modified data from the database – this manual operation reads the browser-edited furniture rotation and location and updates the BIM accordingly.
- SUBSCRIBE: set up an external event to automatically poll for web database changes and immediately update the BIM in real time with no manual intervention at all.




Look at the recording of AU 2013 session DV1736 – *Cloud-Based, Real-Time, Round-Trip, 2D Revit Model Editing on Any Mobile Device* to see this in live action, and check out the implementation and complete source code in the two RoomEditorApp and roomedit GitHub repositories (cf. links in Resources at end).

As said, just like all the projects discussed here, the entire analysis, research, implementation and development process is quite extensively documented in public and full detail.

I just migrated and tested the Revit add-in on Revit 2016, which is the version I am showing here, so you should be all set for trying it out yourself.

## One Software Architecture Possibility to Connect Desktop and Cloud

As said, the room editor presented above consists of two components: a desktop add-in running in Revit, and a CouchDB database server.

We start out with a desktop data source, a cloud-based repository and a client on a mobile device:

- BIM – Building Information Model in Revit with add-in
- Cloud-based data repository in a CouchDB NoSQL database
- 2D rendering on mobile device using HTML, SVG and JavaScript

Real-time editing triggers database and BIM update:

- Graphical room editor on mobile device
- Update cloud database
- Reflect real-time changes in BIM

Due to the same origin policy, the JavaScript implementation uses server-side scripting. This is provided by the CouchDB database, which is a web server at the same time, and reduces the number of components to two instead of three.

## Cloud Database, NoSQL, the CAP Theorem and ACID versus BASE

NoSQL stands for *Not only SQL* and is the main modern database paradigm, a new generation following SQL and transactional, relational databases. It addresses their scalability issues and tends to be non-relational, distributed, open-source, highly scalable and capable of handling huge amounts of data. Frequent other characteristics include schema-free, easy replication support, simple API, eventually consistent.

Traditional transactional databases conform to the ACID paradigm, an acronym for Atomicity, Consistency, Isolation and Durability. These characteristics guarantee that all database transactions are processed reliably and that the database is in a consistent state every time a client accesses it.

Unfortunately for ACID, the CAP Theorem offers a strict mathematical proof that the ACID paradigm cannot simultaneously guarantee consistency, availability and partition tolerance, required for distributed systems.

The modern alternative BASE stands for Basic Availability, Soft-state, Eventual consistency. The system is not guaranteed to be in a consistent state at any given moment. Consistency is guaranteed, eventually.

Most really large web sites today use NoSQL databases to handle large numbers of transaction in a distributed and scalable manner.

## FireRating in the Cloud

Let us look at another, much simpler sample that I implemented earlier this year.

FireRating in the Cloud is a multi-project re-implementation of the well-known FireRating SDK sample. It adds one slight enhancement and modernisation to the original sample, which implements three external commands:

- Create a shared 'Fire Rating' parameter and bind it to the Doors category.
- Export fire rating values for all doors in a project to an external spreadsheet.
- Import the modified values back into the project.

The enhancement is simple:

- Store data for multiple projects to one single cloud-hosted database.

Just like the room editor, FireRating in the Cloud consists of two components, a web server fireratingdb and a Revit add-in FireRatingCloud:

- The web server is a very simple node.js REST server driving a MongoDB NoSQL web database. It has no user interface of its own.
- The Revit add-in implements the same three external commands as the original SDK sample and uses REST to read and write the fire rating values to the cloud-hosted database.

Again, bi-directional data exchange between the Revit BIM and a globally accessible cloud-based web database is enabled.

Although fireratingdb does not implement any user interface of its own, the mongo database can be hosted on a platform that does, e.g., mongolab, again making it usable and the data accessible on any device, in any browser.

I discussed the research and development for this in depth on The 3D Web Coder blog. The FireRatingCloud GitHub repository includes an overview of them all.

I host the node.js web server on Heroku and the MongoDB web database on mongolab for free.

Deploying a GitHub project to Heroku can be totally automated.

Using Mongolab to host the database is more comfortable than keeping it locally.

I can switch back and forth each of these between remote or local deployment by simply setting two Boolean variables.

See for yourself in the live demo how utterly cool it is to play with these components all working together: Heroku, Mongolab, and Revit.

We demonstrate this system up and running live, both locally on the desktop and globally accessible on the web using Heroku and mongolab.

We can also dive into the source code at this point. Let's take a look and see how simple the implementation really is.

## Revit Add-In C# REST Client

Retrieve all doors through a filtered element collector, extract their data and PUT it to web server:

```
foreach( Element e in collector )
{
  Debug.Print( e.Id.IntegerValue.ToString() );

  doorData = new DoorData( e,
    project_id, paramGuid );

  jsonResponse = Util.Put(
    "doors/" + e.UniqueId, doorData );

  Debug.Print( jsonResponse );
}
```

PUT helper method using RestSharp:

```
public static string Put(
  string collection_name_and_id,
  DoorData doorData )
{
  var client = new RestClient( RestApiBaseUrl );

  var request = new RestRequest( _api_version + "/"
    + collection_name_and_id, Method.PUT );

  request.RequestFormat = DataFormat.Json;

  request.AddBody( doorData ); // uses JsonSerializer

  IRestResponse response = client.Execute( request );

  var content = response.Content; // raw content as string

  return content;
}
```

## Node.js MongoDB Web Server

The entire mainline server implementation:

```
var pkg = require( './package.json' );
var express = require('express');
var mongoose = require( 'mongoose' );

var localMongo = false;

if(localMongo) {
  // local database
  var mongo_uri = 'mongodb://localhost/firerating';
} else {
  // mongolab hosted
  var mongo_uri = 'mongodb://revit:revit@ds047742.mongolab.com:47742/firerating';
}

mongoose.connect( mongo_uri );
var db = mongoose.connection;
db.on( 'error', function () {
  var msg = 'unable to connect to database at ';
  throw new Error( msg + mongo_uri );
});

var app = express();

var bodyParser = require( 'body-parser' );
app.use( bodyParser.json({ limit: '1mb' }) );
app.use( bodyParser.urlencoded({ extended: true, limit: '1mb' }) );

require( './model/door' );
require( './routes' )( app );

app.get( '/', function( request, response ) {
  response.send( 'Hello from the cloud-based fire rating '
                + 'database ' + pkg.version + '.\n' );
});

app.set( 'port', process.env.PORT || 3001 );
```

```
var server = app.listen(
  app.get( 'port' ),
  function() {
    console.log( 'Firerating server '
                + pkg.version
                + ' listening at port '
                + server.address().port + ' with '
                + (localMongo?'locally ':'mongolab-')
                + 'hosted mongo db.'); }
);
```

That is the simplest complete desktop and cloud connection sample that I am aware of.

Now let's look at something a bit more complicated.

## The CompHound Component Tracker

I have been working on the CompHound component tracker in the past couple of weeks.

CompHound is similar to and based on the FireRating in the Cloud project, also consisting of a web server driving a mongo database with a REST API invoked by a Revit add-in.

In addition, it sports a user interface for online component usage analysis, reporting, bills of materials, viewing and model navigation.

The Revit add-in is simpler that FireRatingCloud one, since it only implements one-way data writing from the desktop to the cloud.

The web server is more complex, though.

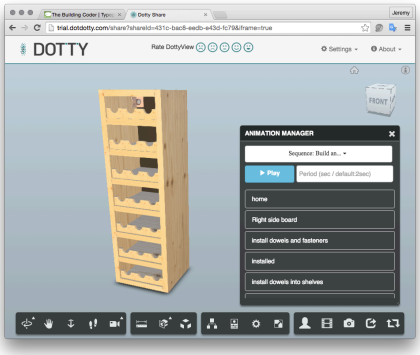
For more details, please refer to the overview of the existing extensive documentation provided by the GitHub repository.

The one and only thing that I will mention is that it also includes a viewer enabling 3D exploration of the component occurrences in situ, using the Autodesk View and Data API.

Here is another compelling example of using that:

## Dotty Animated 3D Assembly Instructions

This one is really self-explanatory.




Check out this compelling sample by Dotty showing how the View and Data API viewer can be used to display animated 3D assembly instructions in the browser.

Can you imagine assembly instructions based on simple 2D images, drawing, words, trying to compete with this?

Think of the end user aspect, reading and understanding static information.

Also consider the preparation side, all the careful analysis, editing and translation required for equivalent text instructions.

You have probably seen the industry and culture moving away from text to printed 2D images with minimal wording... imagine where it will go using animated 3D.

How long will people still be using static assembly instructions printed on paper?

I have even started keeping my personal collection of recipes in a GitHub repository so I can read them more comfortably on an iPad in the kitchen while cooking...

## Conclusion

- Wow, how simple is this!
- Wow, how scalable is this! I can put specific data for all my projects into one single container for global search and analysis.
- Wow, how powerful is this stuff working with interactive 3D models in the browser!
- Wow, how useful is it to be able to share certain data globally in the browser, with anybody I choose, with zero installation, on any mobile device!

I am sure you can find some use for these kinds of opportunities as well.

If you don't, please watch out for those that do...

## Learning More and Further Resources

- Revit Developer Centre: DevTV and My First Plugin Introduction, SDK, Samples, API Help
  http://www.autodesk.com/developrevit
- Developer Guide and Online Help
  http://www.autodesk.com/revitapi-help
- Autodesk Community Revit API Discussion Group
  http://forums.autodesk.com > Revit Architecture > Revit API
- ADN AEC DevBlog
  http://adndevblog.typepad.com/aec
- The Building Coder Revit API Blog
  http://thebuildingcoder.typepad.com
- ADN, The Autodesk Developer Network
  http://www.autodesk.com/joinadn and http://www.autodesk.com/adnopen
- DevHelp Online for ADN members
  http://adn.autodesk.com
- NoSQL: http://nosql-database.org, https://en.wikipedia.org/wiki/NoSQL,
  http://www.mongodb.com/nosql-explained

- GitHub repositories:
    - RoomEditorApp – https://github.com/jeremytammik/RoomEditorApp
    - Roomedit – https://github.com/jeremytammik/roomedit
    - FireRatingCloud – https://github.com/jeremytammik/FireRatingCloud
    - Fireratingdb – https://github.com/jeremytammik/firerating
    - CompHound – https://github.com/CompHound/CompHound.github.io
- Dotty Animated 3D Assembly Instructions
  http://trial.dotdotty.com/share?shareId=431c-bac8-eedb-e43d-fc79&iframe=true
- Connecting desktop and cloud presentation at RTC
  http://thebuildingcoder.typepad.com/blog/2015/11/connecting-desktop-and-cloud-room-editor-update.html
- Connecting desktop and cloud at the AU AEC booth recording
  http://thebuildingcoder.typepad.com/blog/2015/11/connecting-desktop-and-cloud-at-au-and-devdays.html