CORONA

instant application server

Todd Bandrowsky

Country Video Games

# introduction

Corona is a simple integrated application and database server for Windows designed to make it simple to build effective service stacks. It can serve entirely standalone, as its own database and web server, with a single consistent object oriented and rich querying language.   
  
Corona

* Simplify your development with a object oriented web API that can have classes either in its own databases or connected OBBC databases, all from a single schema file.
* Supports out of the box instant B2C experience.
* Internal user communities, such as a company and federated consultants, can have one set of permissions and workflow options, and end customers can have another, just by specifying the configuration.
  + Teams based security.
  + Teams based workflow.
* Create rich applications with near real time analytics embedded into the application workflow.
  + Join, group by, project and filter across any class in the system.
* Easy to develop from any client.

Corona Schema:

* Supports classes, and derived classes
* Supports its own change management, with versioned changes for classes, objects and imports
* Supports imports of delimited files

# command line and configuration

Corona is launched from the command line. It accepts a single parameter indicating the name of its configuration file.

The configuration file looks like this. It contains things like ports, server connections, and importantly, where the schema file lives.

This is from the candidate\_config.json that comes with the system as an example.

The system uses the sendgrid api to send out enrollment emails, users getting a confirmation code.

Connections is a map of names followed by an ODBC connection string.

Both the connections and the SendGrid can be overlaid by using environment variables. This is forced to be upper case to match the convention of upper case environment variables. Use CONNECTION\_SENDGRID for the send grid string, and CONNECTION\_*SOURCENAME* for a sql server connection.

A screenshot of a computer

Description automatically generated

In this example, you would use *CONNECTION\_ADVENTUREWORKS2016* to override the contents of the config file.

{

"SendGrid": {

"ApiKey": "Your key goes here"

},

"Connections": {

"AdventureWorks2016": "Driver={ODBC Driver 17 for SQL Server};Server=BANDROWSKY-RIPP\\DEV01;Database=AdventureWorks2016;Trusted\_Connection=yes;TrustServerCertificate=yes"

},

"Server": {

"listen\_point": "http://localhost:5678/corona",

"application\_name": "coronademo",

"schema\_filename": "candidate\_schema.json",

"database\_filename": "candidate\_database.cdb",

"sys\_user\_name": "system",

"sys\_user\_password": "systempassword",

"sys\_user\_email": "todd.bandrowsky@countryvideogames.com",

"new\_user\_default\_team": "Guests"

}

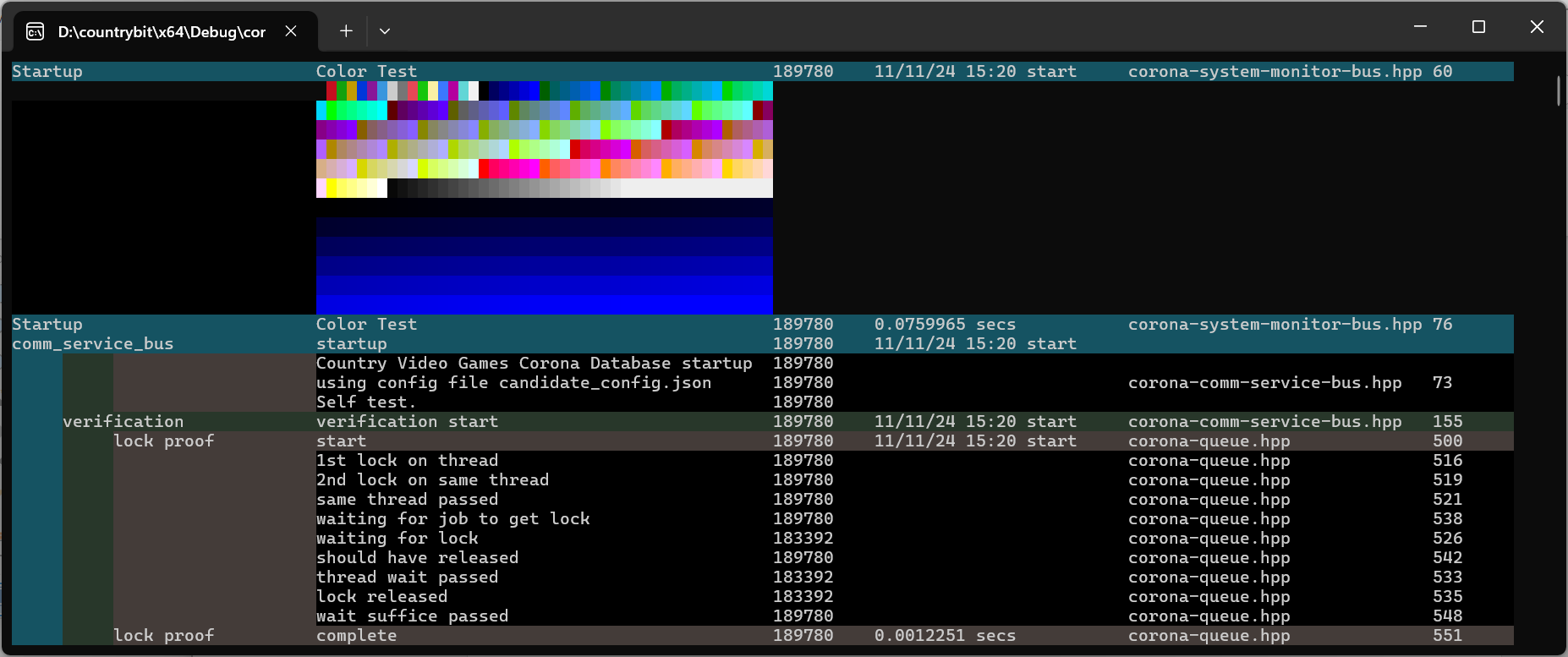
}

When a database is created, the system must create a super user account that is the root of all things. These are created by the sys\_user\* settings above. new\_user\_default\_team isn’t implemented.

Corona creates a single database file, .cdb, for the database specified by a schema. This lives at the file above.   
  
The schema file is applied at start up and during modification.

Once Corona starts, it tests the colors on the terminal, and then,

1. Walks through unit tests.
2. Reads and applies the config.
3. Creates the database, if needed.
4. Works through the schema and applies changes.



A screenshot of a computer

Description automatically generated

Corona can handle requests during startup. https requests are logged to the console. Binding points will be displayed during start up like so.

A screenshot of a computer

Description automatically generated

Additionally, there is a postman collection that you can use to experiment with the server.

# the schema file

The schema file is what defines a Corona application. It describes the classes in a system, and the permissions different teams can have to use them.

A screenshot of a computer

Description automatically generated

A schema file has a name, description and version, along with author credits, and then has three main sections, *classes, users* and *datasets.*

## teams

Corona extends the concept of a security role into a team. A team has security permissions, like a role does, but it also defines workflow. Quite simply, a team is a collection of people doing the same kind of job, with the tools for those people that do that job.

## classes

Classes are the heart of Corona. A class is a Corona is basically an object schema bound to a store. A class has:

* Name and Description – class\_name and class\_description.
* Base Class – base\_class\_name gives the name of a base class. You must have a base class that is sys\_object, or your own class. This is because sys\_object contains the object\_id, which is global to the database, and tracks the user names and dates of object modifications.
* Fields – a map of fields, consisting of field names to simple description.
* Indexes – names of indexes
* Sql – specifies mappings to an existing SQL source.

Simple classes

A computer screen shot of text

Description automatically generated

Core class properties

|  |  |
| --- | --- |
| class\_name | Unique name of class |
| class\_description | Human description |
| base\_class\_name | Either use *sys\_object* as a base class, or your own. Single inheritance only. |

### fields

A Corona class can have fields. There’s ways to specify a field. The quick way is a field name mapped to one of several types:

A screen shot of a computer code

Description automatically generated

Corona Field Types

|  |  |
| --- | --- |
| string | A string field |
| int64 | A 64 bit integer, useful for ids |
| double | A number |
| datetime | A datetime |
| array | An array, in the JSON sense |
| object | An object, in the JSON sense |
| query | Query based on using members of the object as parameters. |
| drop\_down | A scalar with a class for a source. For lookups |
|  |  |

Additionally, each field type can have an extended specification.

Instead of:

“my\_field” : “string”,

One might do

“my\_field”: {

“field\_type”:”string”

}

These extended options are useful for validation. Corona automatically validates when an object is put.

Extended options for strings.

|  |  |
| --- | --- |
| min\_length | An integer indicating the minimum length of a string |
| max\_length | An integer indicating the maximum length of a string |
| match\_pattern | A regular expression that the string must match |
| enum | A list of strings, that the string must be one of. Case insensitive. |
|  |  |

Extended options for int64, double and datetime

|  |  |
| --- | --- |
| min\_value | Minimum value of field |
| max\_value | Maximum value of field |

Extended options for arrays and objects

|  |  |
| --- | --- |
| child\_objects | Specifies child classes that may be an object member, or an object element of an array field. |

### arrays, objects and child objects

By default, Corona lets you put anything into an object and an array. But Arrays and objects can be constrained by class, and if so, Corona will break out the objects into parent and child tables and keep track of the mappings for you. Conceptually, to do this, you need to have a constructor for a new child, and an assignment for an existing child. In this way, if you have an inbound json with children, Corona classes will know where to put them.

The structure you need to create to get this effect is “child\_objects”.

A child objects looks like this. Here we have a “permissions” field, which is an array. On that array, there is a child\_objects which allows the contents of the array to be one of the types specified, or a derived class.

A computer screen shot of text

Description automatically generated

|  |  |
| --- | --- |
| child\_class\_name | The base class name of the child object. |
| copy\_values | A map of from -> to values. When a child object is placed onto the collection, corona copies the elements in the copy\_values from the parent to the child. |
| construct\_values | A map of from -> to values. When a child object is created onto the collection, corona copies the elements in the copy\_values from the parent to the child. |

By default, Corona does *not* return child objects. But if you use the get method, you can include\_children to get child objects.

A screenshot of a computer

Description automatically generated

Also, you can always query a child class directly, similar to how you might do it in a relational database.

A screenshot of a computer program

Description automatically generated

## sql mappings

## datasets

Corona schema files support the concept of datasets for load out. These are versioned in a fashion similar to Liquibase, with the developer intent that reapplying a schema won’t cause a dataset to be reapplied if the version is the same.

A DataSet can be one of two things. It can have a list of objects which are put immediately, and it can have a CSV file for import.

### objects

Objects are specified in a data set. The objects in a dataset is just an array of objects, with some header information to identify it.

A screen shot of a computer program

Description automatically generated

The objects themselves, are just objects in json, just like they would be put. For example, look at our teams again.

A computer screen with text

Description automatically generated

### imports

An import data set looks like this. Presently only csv is supported, which is a bare delimited file. The delimiter may be specified. The filename given is the source filename for the import, and the target\_class is where it goes. A column\_map maps the field on the right to the column id on the left.

A screen shot of a computer

Description automatically generated

Corona processes the schema import csvs in sets of 1000.

Issues with malformed objects are reported.

A screen shot of a computer

Description automatically generated

# json api

## introduction

Corona uses a Json Api. The schema is ultimately the source for the api, but the structure of the api is the same regardless of what you put in it. In this way, Corona lets you think about your data, and not worry about the details of converting json back and forth, serialization and all of that. Corona just does it.

### / corona/login/loginuser

You can login through the system login you supplied through the config file. And you can log in with users that you create:

A screenshot of a computer

Description automatically generated

If the login is successful, you will receive a token that you can supply to headers for future activities. This token expires, at which point, you’ll have to login again. In PostMan, this looks like this:

A close up of a computer screen

Description automatically generated

### / corona/login/createuser

Creates an unconfirmed user in the system. The user is sent an confirmation to the email address, and /corona/login/confirmuser must be called with that code.

Elements

|  |  |  |
| --- | --- | --- |
| data |  |  |
|  | first\_name | String first name |
|  | last\_name | String last name |
|  | user\_name | Mandatory user name. if duplicated, a different one will be returned. Use that one. |
|  | email | The email where the user will receive their account confirmation code |
|  | mobile | Cell phone number. In future releases we will use MFA |
|  | street1 | The street address, etc, are all encouraged |
|  | street2 |  |
|  | city |  |
|  | state |  |
|  | zip |  |
|  | password1 | Passwords have to match on submission and they are encrypted. |
|  | password2 |  |