# D IS FOR (DE)SERIALIZATION

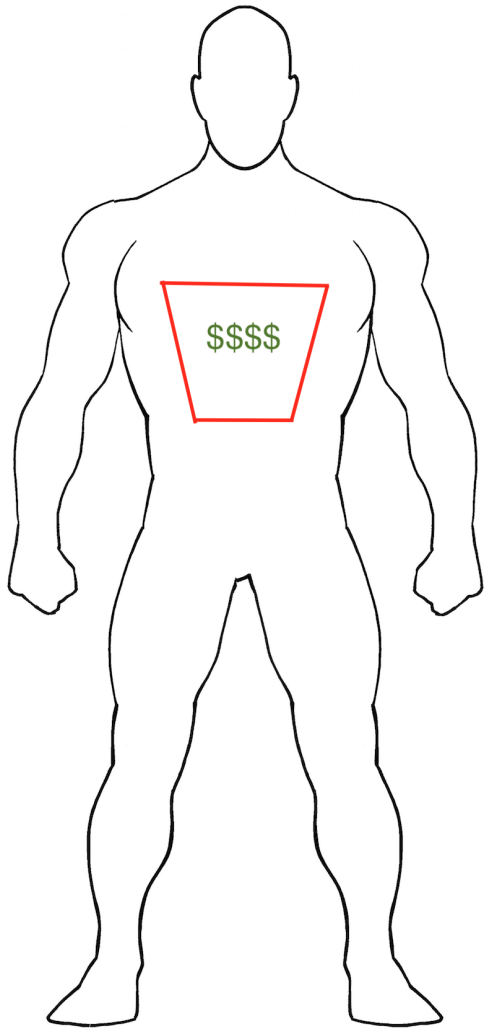
STEVEN SCHVEIGHOFFER

# PAYROLL!!!

# PAYROLL!!!

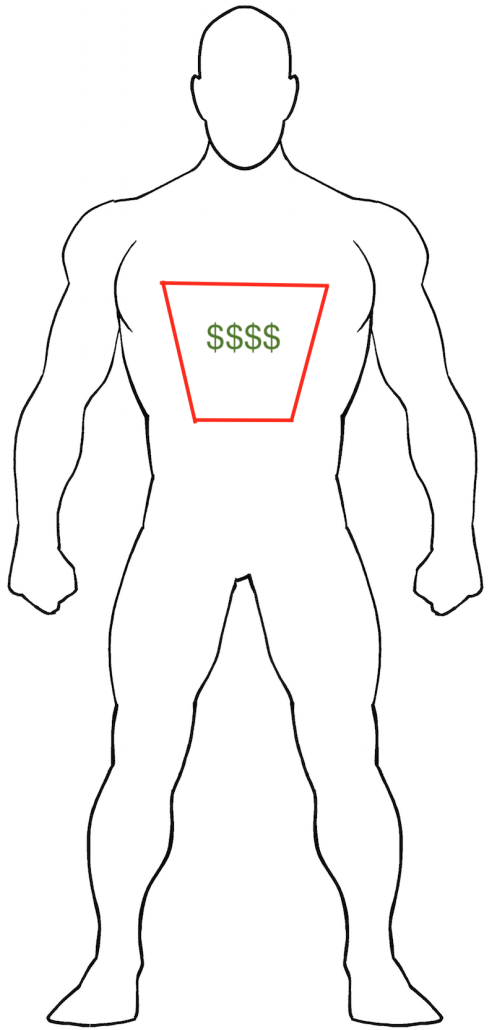
- In the fall of 2016, our HR manager gave her notice.
- Needed someone to handle Payroll
- Who could do such a task?

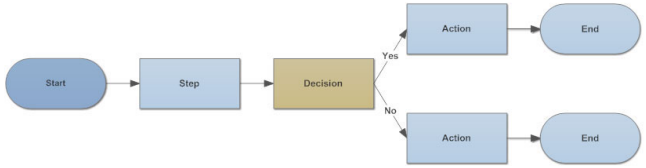
# PAYROLL MAN!



# PAYROLL MAN!

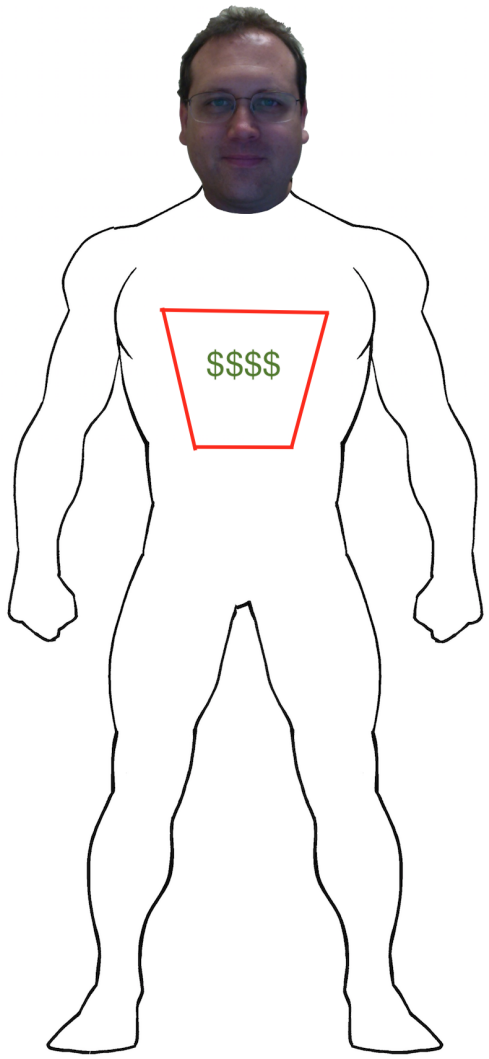
2 + 2 = 4

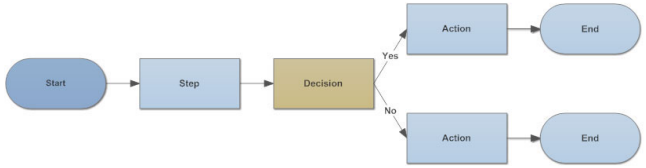


# PAYROLL MAN!

2 + 2 = 4



# STEPS FOR PAYROLL

# 1. PAYROLL TIMESHEET

- Print one at the start of every payroll
- Sorted intuitively by department
- Record hours
- Record changes to status

# TIMESHEET

*Payroll Specialist: Michael*

| PERSONNEL INFORMATION | RATE(S) | REGULAR HOURS | OVERTIME HOURS | OTHER (Indicate Hours or $) | EARNINGS (E) / DEDUCTIONS (D) | YEAR TO DATE |
|---|---|---|---|---|---|---|
| [redacted] | Wks Worked 46<br>Bi-weekly<br><br>YTD Hours Used: S 0.00/V 0.00<br>YTD Wages & Tips: 46,554.26 | 23.1000/Hour | 40 | 9.07 | PTO - 24<br>Hol - 16 | |
| | Wks Worked 46<br>Bi-weekly<br><br>YTD Hours Used: S 0.00/V 0.00<br>YTD Wages & Tips: 46,017.71 | (1) 2,000.77/Pay Period<br>(2) 25.0100/Hour<br>Raise to: | Pay Salary ☐<br>$53,600 - | ✓ 2061.53 - | | |
| | Wks Worked 46<br>Bi-weekly<br><br>YTD Hours Used: S 0.00/V 0.00<br><br>YTD Wages & Tips: 38,115.41 | 20.7100/Hour | 60.45 | 2.19 | Hol - 16 | |
| | Wks Worked 46<br>Bi-weekly<br><br>YTD Hours Used: S 0.00/V 0.00<br>YTD Wages & Tips: 33,133.47 | 18.5000/Hour | 62.54 | .19 | Hol - 16 | |
| | Wks Worked 46<br>Bi-weekly<br><br>YTD Hours Used: S 0.00/V 0.00<br>YTD Wages & Tips: 99,028.11 | 4,305.57/Pay Period<br>Raise → | Pay Salary ☐<br>$116,425. - | 4477.88 | | |
| | Wks Worked 46<br>Bi-weekly<br><br>YTD Hours Used: S 0.00/V 0.00<br>YTD Wages & Tips: 71,726.85 | (1) 2,673.08/Pay Period<br>(2) 33.4100/Hour | Pay Salary ☐ | | | Terminate - stop salary!! |
| | Wks Worked 46<br>Bi-weekly<br><br>YTD Hours Used: S 0.00/V 0.00<br>YTD Wages & Tips: 65,758.74 | (1) 1,778.85/Pay Period<br>(2) 22.2400/Hour | Pay Salary ☐ | | | Comm: 1399.05 |
| | Wks Worked 20<br>Bi-weekly<br><br>YTD Hours Used: S 0.00/V 0.00<br>YTD Wages & Tips: 23,948.00 | 32.0000/Hour | 61 | | Hol - 16 | |
| | Wks Worked 46<br>Bi-weekly<br><br>YTD Hours Used: S 0.00/V 0.00<br>YTD Wages & Tips: 47,305.53 | 25.2500/Hour | 61 | 8 | Hol - 16 | |
| | Wks Worked 46<br>Bi-weekly<br><br>YTD Hours Used: S 0.00/V 0.00<br>YTD Wages & Tips: 52,474.13 | (1) 2,307.69/Pay Period<br>(2) 20.6700/Hour | Pay Salary ☐ | | | Comm: 185.55 |

# 2. INDIVIDUAL TIMESHEETS

- Time started and stopped each task
- Hours for each task
- Total hours for day
- Drive time (am/pm)
- Total hours for week (regular and overtime)
- Overnights

Calendar Week 2

"Extra Drive Time" is amount of time over 1 hour commute to and from job site. "Total Hours"= (Extra Drive Time AM and PM + Total Hrs Job 1, 2, 3 4)

Reg.=Hours paid at your normal hourly rate. PW=Hours paid at prevailing wage rate.

NRM

Payroll Time Record for Period: **1/8/2017** **Thru** **1/14/2017** Pay Period: 1 Week: 2

| | Extra Drive Time AM | Arrive Job 1 | Leave Job 1 | Hours Job 1 | Start Job 2 | Leave Job 2 | Hours Job 2 | Start Job 3 | Leave Job 3 | Hours Job 3 | Start Job 4 | Leave Job 4 | Hours Job 4 | Extra Drive Time PM | Straight Time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sunday 1/8/17 | | | | | | | | | | | | | | | Reg / PW |
| | | Job 1# | | | Job 2# | | | Job 3# | | | Job 4# | | | | |
| Monday 1/9/17 | | 8:00 | 10:00 | 2 | 4:00 | 7:30 | 3.5 | | | | | | | | 5.5 Reg |
| | | Job 1# 55 (PREPARE FOR JAMES) | | | Job 2# PICKING JAMES UP FROM AIRPORT | | | Job 3# | | | Job 4# | | | | |
| Tuesday 1/10/17 | | 7:30 | 8:30 | 1 | 8:30 | 9:00 | .5 | 9:00 | 3:30 | 6.5 | | | | | 7.5 Reg |
| | | Job 1# 55 | | | Job 2# 99 | | | Job 3# NC-17120-NRM | | | Job 4# | | | | |
| Wednesday 1/11/17 | | 8:00 | 10:30 | 2.5 | 10:30 | 11:00 | .5 | 11:00 | 4:30 | 5.5 | | | | | 8 Reg |
| | | Job 1# NC-21840-NRM | | | Job 2# 99 | | | Job 3# NC-16799-NRM | | | Job 4# | | | | |
| Thursday 1/12/17 | | 8:00 | 2:00 | 6 | 2:00 | 2:30 | .5 | 2:30 | 4:30 | 2 | | | | | 8.5 Reg |
| | | Job 1# NC-16799-NRM | | | Job 2# 99 | | | Job 3# (VAN-OIC CHANGE) | | | Job 4# | | | | |
| Friday 1/13/17 | | 8:00 | 5:30 | 9.5 | | | | | | | | | | | 10 Reg |
| | | Job 1# 25581 | | | Job 2# | | | Job 3# | | | Job 4# | | | | |
| Saturday 1/14/17 | | | | | | | | | | | | | | | Reg / PW |
| | | Job 1# | | | Job 2# | | | Job 3# | | | Job 4# | | | | |

Print Name | Employee Signature | 1/13/17 Date

Number of Overnights: ____

On Call?: ____

| | Straight |
|---|---|
| Week Regular Hours: | 39.5 |
| Week PW Hours: | |
| Total PTO Hours: | |

Email time sheets to lmatthews@nrminc.com and copy your supervisor.

All hours must be assigned to a Job # or WO #. Other time assigned as follows:

99=Drive Time / 55=Warehouse / 44=Office

Calenc[illegible] Week 5

"Extra Drive Time" is amount of time over 1 hour commute to and from job site. "Total Hours"= (Extra Drive Time AM and PM + Total Hrs Job 1, 2, 3 4)
Reg.=Hours paid at your normal hourly rate. PW=Hours paid at prevailing wage rate.

NRM

Payroll Time Record for Period: **1/29/2017** **Thru** **2/4/2017** Pay Period: 3
Week: 1

| | Extra Drive Time AM | Arrive Job 1 | Leave Job 1 | Hours Job 1 | Start Job 2 | Leave Job 2 | Hours Job 2 | Start Job 3 | Leave Job 3 | Hours Job 3 | Start Job 4 | Leave Job 4 | Hours Job 4 | Extra Drive Time PM | Straight Time | Over Time Hours |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sunday 1/29/17 | | | | | | | | | | | | | | | Reg. | Reg. |
| | | Job 1# | | | Job 2# | | | Job 3# | | | Job 4# | | | | PW | PW |
| Monday 1/30/17 | (6.5) | 1:00 | 3:30 | (2.5) | 3:30 | 5:00 | (1.5) | | | | | | | | 9 Reg. | 1.5 Reg. |
| | | Job 1# | Z6395-RM | | Job 2# | 55 | | Job 3# | | | Job 4# | | | | PW | PW |
| Tuesday 1/31/17 | | 8:00 | 3:00 | (7) | | | | | | | | | | | 7 Reg. | Reg. |
| | | Job 1# | Z6395-RM | | Job 2# | | | Job 3# | | | Job 4# | | | | PW | PW |
| Wednesday 2/1/17 | | 7:00 | 9:00 | (2) | 9:00 | 5:00 | (8) | | | | | | | (1) | 8 Reg. | 3 Reg. |
| | | Job 1# | 55 | | Job 2# | Z5097-RM | | Job 3# | | | Job 4# | | | | PW | PW |
| Thursday 2/2/17 | (2) | 8:00 | 12:00 | (4) | 12:00 | 2:30 | (2.5) | 2:30 | 3:30 | (1) | | | | (1) | 8 Reg. | 2.5 Reg. |
| | | Job 1# | Z5714-TF | | Job 2# | WO# Z1247 | | Job 3# | WO# Z1280 | | Job 4# | | | | PW | PW |
| Friday 2/3/17 | | 7:00 | 9:00 | (2) | 9:00 | 10:30 | (1.5) | 10:30 | 12:30 | (2) | | | | (5.5) | 8 Reg. | 3 Reg. |
| | | Job 1# | WO# Z1183 | | Job 2# | WO# Z1178 | | Job 3# | WO# Z1181 | | Job 4# | | | | PW | PW |
| Saturday 2/4/17 | | | | | | | | | | | | | | | Reg. | Reg. |
| | | Job 1# | | | Job 2# | | | Job 3# | | | Job 4# | | | | PW | PW |

| | Straight | OT |
|---|---|---|
| Week Regular Hours: | 40 | 10 |
| Week PW Hours: | | |
| Total PTO Hours: | | |
| Total Holiday Hours: | | |

Number of Overnights: (4)
On Call?:

Print Name — Employee Signature — Date: 2/4/17

Supervisor Signature

Email time sheets to lmatthews@nrminc.com and copy your supervisor.
All hours must be assigned to a Job # or WO #. Other time assigned as follows:
99=Drive Time / 55=Warehouse / 44=Office

# 3. PUNCHCARDS

- Office employees use RFID punchcards
- But must also manually write them down
- Shore up punch card log with written data

# 5. ENTER DATA

- Copy all data from the payroll timesheet to the web site
- Double check all numbers by adding on a calculator!
- Adjust anything that needs adjusting (salary, tax witholdings, etc)

# 6. OTHER STUFF

- Commissions
- Time off report
- Run post-payroll reports
- Print everything, put in a folder

# 6. OTHER STUFF

- Separate folder for each PW job
- Extract 401k report from payroll system
- Update 401k CSV file... BY HAND
- 401k loan updates

# 6. OH GOD IT KEEPS GOING

- Print 401k report for separate folder
- Send commission sheets to salespeople
- Certified payroll
- Print out timesheet for next payroll

# TIME NEEDED TO PREPARE PAYROLL

3-4 days

# TIME AVAILABLE TO PREPARE PAYROLL

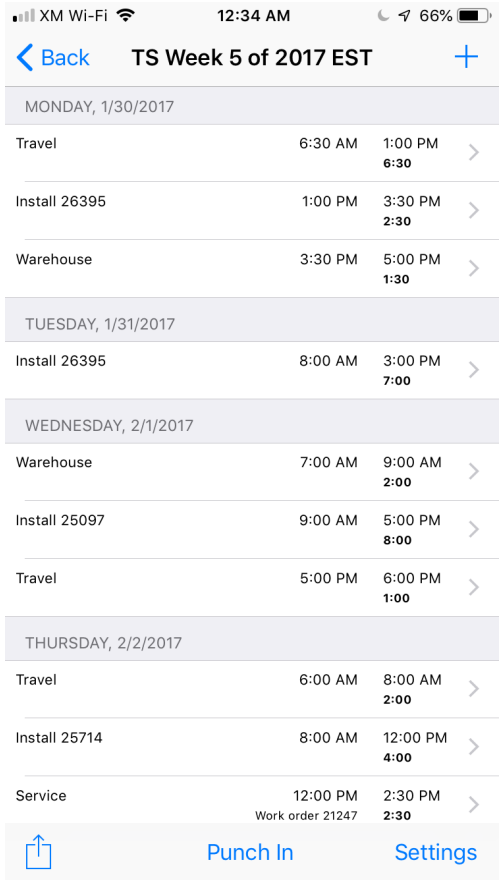
2 days

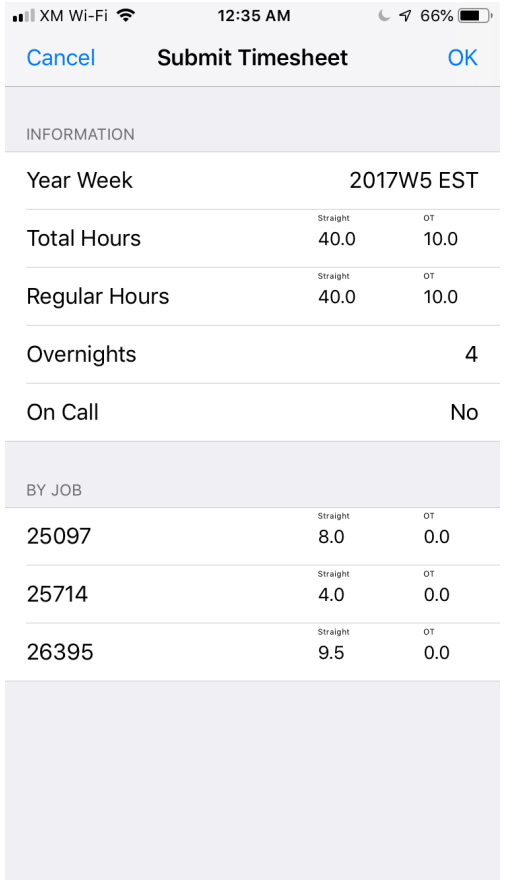
# THE MORAL

"You have a computer, why not use it?"

Use computers for everything they are good at

# TIMECARD APP




- No more time math
- Always have it with them
- Links to jobs in our tracking system
- Replace unreliable punchcard system

# PAYROLL TIMESHEET

- Use Excel to track everything instead of paper
- VBA macro to generate CSV for upload to payroll system
- No more clunky transfer, one button push
- All totals calculated instantly.

# TIME TO RUN PAYROLL NOW?

*4 hours*

# D SERIALIZATION

You have a computer, why not use it?

# D SERIALIZATION

You have a ~~computer~~ D Compiler, why not use it?

# A Simple Example

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    result.id = r.getAsInteger("id");
    result.name = r.getAsString("mane");
    return result;
}
```

# A Simple Example

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    result.id = r.getAsInteger("id");
    result.name = r.getAsString("mane");
    return result;
}
```

# A Simple Example

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    result.id = r.getAsInteger("id");
    result.name = r.getAsString("mane");
    return result;
}
```

Initialize Record

# A Simple Example

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    result.id = r.getAsInteger("id");
    result.name = r.getAsString("mane");
    return result;
}
```

Parse integer field

# A Simple Example

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    result.id = r.getAsInteger("id");
    result.name = r.getAsString("mane");
    return result;
}
```

Parse string field

# A Simple Example

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    result.id = r.getAsInteger("id");
    result.name = r.getAsString("mane");
    return result;
}
```

Return

# A Simple Example

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    result.id = r.getAsInteger("id");
    result.name = r.getAsString("mane");
    return result;
}
```

Does it work?

# A GENERATIVE APPROACH

- `static foreach`
- `__traits/std.traits`
- `static if`

# A Generative Approach

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
   Record result;
   import std.traits;
   static foreach(i, m; FieldNameTuple!Record)
   {
      __traits(getMember, result, m) =
          r.getWithType!(typeof(__traits(getMember, result, m)))(m);
   }
}
```

# A Generative Approach

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
   Record result;
   import std.traits;
   static foreach(i, m; FieldNameTuple!Record)
   {
      __traits(getMember, result, m) =
          r.getWithType!(typeof(__traits(getMember, result, m)))(m);
   }
}
```

Same definition

# A Generative Approach

```
import std.traits;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    import std.traits;
    static foreach(i, m; FieldNameTuple!Record)
    {
        __traits(getMember, result, m) =
            r.getWithType!(typeof(__traits(getMember, result, m)))(m);
    }
}
```

Initialize Record

# A Generative Approach

```d
struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    import std.traits;
    static foreach(i, m; FieldNameTuple!Record)
    {
        __traits(getMember, result, m) =
            r.getWithType!(typeof(__traits(getMember, result, m)))(m);
    }
}
```

Import some generative magic

# A Generative Approach

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    import std.traits;
    static foreach(i, m; FieldNameTuple!Record)
    {
        __traits(getMember, result, m) =
            r.getWithType!(typeof(__traits(getMember, result, m)))(m);
    }
}
```

Get all the field member names

# A Generative Approach

```
struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    import std.traits;
    static foreach(i, m; FieldNameTuple!Record)
    {
        __traits(getMember, result, m) =
            r.getWithType!(typeof(__traits(getMember, result, m)))(m);
    }
}
```

Assign the field to...

# A Generative Approach

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    import std.traits;
    static foreach(i, m; FieldNameTuple!Record)
    {
        __traits(getMember, result, m) =
            r.getWithType!(typeof(__traits(getMember, result, m)))(m);
    }
}
```

The exact conversion expected

# A Generative Approach

```
import db.api;

struct Record
{
    int id;
    string name;
}

Record processRecord(Row r)
{
    Record result;
    import std.traits;
    static foreach(i, m; FieldNameTuple!Record)
    {
        __traits(getMember, result, m) =
            r.getWithType!(typeof(__traits(getMember, result, m)))(m);
    }
}
```

The whole function

# A More complicated case

```
import db.api;

enum RecordType
{
    person,
    location
}

enum Permission
{
    admin = 0x01,
    restricted = 0x02,
    guest = 0x04,
}

struct Point
{
```

# A More complicated case

```
import db.api;

enum RecordType
{
    person,
    location
}

enum Permission
{
    admin = 0x01,
    restricted = 0x02,
    guest = 0x04,
}

struct Point
{
```

Enums and Types

# A More complicated case

```
{
    admin = 0x01,
    restricted = 0x02,
    guest = 0x04,
}

struct Point
{
    int y;
    int x;
}

struct Record
{
    int id;
    string name;
    RecordType type; // stored as integer
```

Enums and Types

# A More complicated case

```
    int y;
    int x;
}

struct Record
{
    int id;
    string name;
    RecordType type; // stored as integer
    Point location; // location_x, location_y, only present if type is location
    string comment; // optional
    Permission permissions; // stored as comma-separated string list
    int _class; // really called "class" in the DB
}

Record processRecord(Row r)
{
    Record result;
```

More complex, more convenient

# A More complicated case

```
    String name;
    RecordType type; // stored as integer
    Point location; // location_x, location_y, only present if type is location
    string comment; // optional
    Permission permissions; // stored as comma-separated string list
    int _class; // really called "class" in the DB
}

Record processRecord(Row r)
{
    Record result;
    result.id = r.getAsInteger("id");
    result.name = r.getAsString("name"); // typo corrected!
    // Record type is stored as an integer, so use a direct cast
    result.type = cast(RecordType)r.getAsInteger("type");
    // location is stored as 2 fields, location_x, and location_y
    if(type == RecordType.location)
        location = Point(r.getAsInteger("location_x"),
```

Declare the result

# A More complicated case

```
struct Record
{
    int id;
    string name;
    RecordType type; // stored as integer
    Point location; // location_x, location_y, only present if type is location
    string comment; // optional
    Permission permissions; // stored as comma-separated string list
    int _class; // really called "class" in the DB
}

Record processRecord(Row r)
{
    Record result;
    result.id = r.getAsInteger("id");
    result.name = r.getAsString("name"); // typo corrected!
    // Record type is stored as an integer, so use a direct cast
    result.type = cast(RecordType)r.getAsInteger("type");
```

Same as before

# A More complicated case

```
    int id;
    string name;
    RecordType type; // stored as integer
    Point location; // location_x, location_y, only present if type is location
    string comment; // optional
    Permission permissions; // stored as comma-separated string list
    int _class; // really called "class" in the DB
}

Record processRecord(Row r)
{
    Record result;
    result.id = r.getAsInteger("id");
    result.name = r.getAsString("name"); // typo corrected!
    // Record type is stored as an integer, so use a direct cast
    result.type = cast(RecordType)r.getAsInteger("type");
    // location is stored as 2 fields, location_x, and location_y
    if(type == RecordType.location)
```

Need to parse this as an integer

# A More complicated case

```
    int id;
    string name;
    RecordType type; // stored as integer
    Point location; // location_x, location_y, only present if type is location
    ...
}

Record processRecord(Row r)
{
    ...
    // Record type is stored as an integer, so use a direct cast
    result.type = cast(RecordType)r.getAsInteger("type");
    // location is stored as 2 fields, location_x, and location_y
    if(type == RecordType.location)
        location = Point(r.getAsInteger("location_x"),
                         r.getAsInteger("location_y"));
    // comment is optional, avoid exception for NULL storage
```

Parse as location_x, location_y

# A More complicated case

```
    string name;
    RecordType type; // stored as integer
    Point location; // location_x, location_y, only present if type is location
    string comment; // optional
    ...
}

Record processRecord(Row r)
{
    ...
    if(type == RecordType.location)
        location = Point(r.getAsInteger("location_x"),
                         r.getAsInteger("location_y"));
    // comment is optional, avoid exception for NULL storage
    if(!r.fieldIsNull("comment"))
        result.comment = r.getAsString("comment");
```

Only store comment if present

# A More complicated case

```
    string name;
    RecordType type; // stored as integer
    Point location; // location_x, location_y, only present if type is location
    string comment; // optional
    Permission permissions; // stored as comma-separated string list
    int _class; // really called "class" in the DB
}

Record processRecord(Row r)
{
    ...
    // permissions is stored as a comma-separated string list.
    result.permissions = r.getAsString("permissions")
        .splitter(',')
        .map!(s => s.to!Permission)
        .reduce!((a, b) => cast(Permission)(a|b));
    result._class = r.getAsInteger("class");
```

Comma-seperated map parsing

# A More complicated case

```
    string name;
    RecordType type; // stored as integer
    Point location; // location_x, location_y, only present if type is location
    string comment; // optional
    Permission permissions; // stored as comma-separated string list
    int _class; // really called "class" in the DB
}

Record processRecord(Row r)
{
    ...
    // permissions is stored as a comma-separated string list.
    result.permissions = r.getAsString("permissions")
        .splitter(',')
        .map!(s => s.to!Permission)
        .reduce!((a, b) => cast(Permission)(a|b));
    result._class = r.getAsInteger("class");
```

avoid keyword collision

```
    result.name = r.getAsString("name"); // typo corrected!
    // Record type is stored as an integer, so use a direct cast
    result.type = cast(RecordType)r.getAsInteger("type");
    // location is stored as 2 fields, location_x, and location_y
    if(type == RecordType.location)
        location = Point(r.getAsInteger("location_x"),
                         r.getAsInteger("location_y"));
    // comment is optional, avoid exception for NULL storage
    if(r.hasField("comment") && !r.fieldIsNull("comment"))
        result.comment = r.getAsString("comment");

    // permissions is stored as a comma-separated string list.
    result.permissions = r.getAsString("permissions")
        .splitter(',')
        .map!(s => s.to!Permission)
        .reduce!((a, b) => cast(Permission)(a|b));
    result._class = r.getAsInteger("class");
```

Can you spot the bug?

```
struct Point
{
    int y;
    int x;
}

...


Record processRecord(Row r)
{
    ...
    // location is stored as 2 fields, location_x, and location_y
    if(type == RecordType.location)
        location = Point(r.getAsInteger("location_x"),
                         r.getAsInteger("location_y"));
    // comment is optional, avoid exception for NULL storage
```

How about now?

# MORE GENERATIVE APPROACH

- User Data Attributes (UDAs) instead of comments
- Compiler can react to them, just like we react to the comments.

# Replace comments with UDAs

```
struct Record
{
    int id;
    string name;
    RecordType type; // stored as integer
    Point location; // stored as location_x, location_y, only
    string comment; // optional
    Permission permissions; // stored as comma-separated strin
    int _class; // really called "class" in the DB
}
```

# Replace comments with UDAs

```
struct Record
{
    int id;
    string name;
    RecordType type; // stored as integer
    Point location; // stored as location_x, location_y, only
    string comment; // optional
    Permission permissions; // stored as comma-separated strin
    int _class; // really called "class" in the DB
}
```

# Replace comments with UDAs

```
struct Record
{
    int id;
    string name;

    @dbType!int
    RecordType type;

    Point location; // stored as location_x, location_y, only
    string comment; // optional
    Permission permissions; // stored as comma-separated strin
    int _class; // really called "class" in the DB
}
```

You can pass types to UDAs

# Replace comments with UDAs

```
struct Record
{
    int id;
    string name;

    @dbType!int
    RecordType type;

    Point location; // stored as location_x, location_y, only
    string comment; // optional
    Permission permissions; // stored as comma-separated strin
    int _class; // really called "class" in the DB
}
```

# Replace comments with UDAs

```
struct Record
{
    int id;
    string name;

    @dbType!int
    RecordType type;

    @dbOnlyIf!(R => r.type == RecordType.location)
    Point location;

    string comment; // optional
    Permission permissions; // stored as comma-separated strin
    int _class; // really called "class" in the DB
}
```

And lambdas

# Replace comments with UDAs

```
struct Record
{
    int id;
    string name;

    @dbType!int
    RecordType type;

    @dbOnlyIf!(R => r.type == RecordType.location)
    Point location;

    string comment; // optional
    Permission permissions; // stored as comma-separated strin
    int _class; // really called "class" in the DB
}
```

# Replace comments with UDAs

```
    int id;
    string name;

    @dbType!int
    RecordType type;

    @dbOnlyIf!(R => r.type == RecordType.location)
    Point location;

    @dbOptional
    string comment;

    Permission permissions; // stored as comma-separated strin
    int _class; // really called "class" in the DB
}
```

Simple tag

# Replace comments with UDAs

```
    int id;
    string name;

    @dbType!int
    RecordType type;

    @dbOnlyIf!(R => r.type == RecordType.location)
    Point location;

    @dbOptional
    string comment;

    Permission permissions; // stored as comma-separated strin
    int _class; // really called "class" in the DB
}
```

# Replace comments with UDAs

```
@dbType!int
RecordType type;

@dbOnlyIf!(R => r.type == RecordType.location)
Point location;

@dbOptional
string comment;

@dbProcessWith!(parseCommaList)
Permission permissions;

int _class; // really called "class" in the DB
```

Pass a normal function

# Replace comments with UDAs

```
    @dbType!int
    RecordType type;

    @dbOnlyIf!(R => r.type == RecordType.location)
    Point location;

    @dbOptional
    string comment;

    @dbProcessWith!(parseCommaList)
    Permission permissions;

    int _class; // really called "class" in the DB
}
```

# Replace comments with UDAs

```
    @dbType!int
    RecordType type;

    @dbOnlyIf!(R => r.type == RecordType.location)
    Point location;

    @dbOptional
    string comment;

    @dbProcessWith!(parseCommaList)
    Permission permissions;

    @dbName("class")
    int _class;
```

and so on...

# Replace comments with UDAs

```
struct Record
{
    int id;
    string name;

    @dbType!int
    RecordType type;

    @dbOnlyIf!(R => r.type == RecordType.location)
    Point location;

    @dbOptional
    string comment;

    @dbProcessWith!(parseCommaList)
    Permission permissions;

    @dbName("class")
    int _class;
}
```

The result

# Generative serialization

```
Record processRecord(Row r)
{
    import std.traits;

    Record result;
    foreach(m; __traits(allMembers, Record))
    {
        static if(hasUDA!(__traits(getMember, Record, m), DbOptional))
        {
            if(!r.hasField(m) || r.fieldIsNull(m))
                continue;
        }
        static if(hasUDA!(__traits(getMember, Record, m), DbOnlyIf))
        {
            if(!getUDA!(__traits(getMember, Record, m), DbOnlyIf).pred(result))
                continue;
        }
```

# Generative serialization

```
Record processRecord(Row r)
{
    import std.traits;

    Record result;
    foreach(m; __traits(allMembers, Record))
    {
        static if(hasUDA!(__traits(getMember, Record, m), DbOptional))
        {
            if(!r.hasField(m) || r.fieldIsNull(m))
                continue;
        }
        static if(hasUDA!(__traits(getMember, Record, m), DbOnlyIf))
        {
            if(!getUDA!(__traits(getMember, Record, m), DbOnlyIf).pred(result))
                continue;
        }
```

Begins the same as our other functions

# Generative serialization

```
Record processRecord(Row r)
{
    import std.traits;

    Record result;
    foreach(m; __traits(allMembers, Record))
    {
        static if(hasUDA!(__traits(getMember, Record, m), DbOptional))
        {
            if(!r.hasField(m) || r.fieldIsNull(m))
                continue;
        }
        static if(hasUDA!(__traits(getMember, Record, m), DbOnlyIf))
        {
            if(!getUDA!(__traits(getMember, Record, m), DbOnlyIf).pred(result))
                continue;
        }
```

use NORMAL foreach?

# Generative serialization

```
{
    import std.traits;

    Record result;
    foreach(m; __traits(allMembers, Record))
    {
        static if(hasUDA!(__traits(getMember, Record, m), DbOptional))
        {
            if(!r.hasField(m) || r.fieldIsNull(m))
                continue;
        }
        static if(hasUDA!(__traits(getMember, Record, m), DbOnlyIf))
        {
            if(!getUDA!(__traits(getMember, Record, m), DbOnlyIf).pred(result))
                continue;
        }
        static if(hasUDA!(__traits(getMember, Record, m), DbProcessWith))
```

skip members if they are optional and not present

# Generative serialization

```
    {
        static if(hasUDA!(__traits(getMember, Record, m), DbOptional))
        {
            if(!r.hasField(m) || r.fieldIsNull(m))
                continue;
        }
        static if(hasUDA!(__traits(getMember, Record, m), DbOnlyIf))
        {
            if(!getUDA!(__traits(getMember, Record, m), DbOnlyIf).pred(result))
                continue;
        }
        static if(hasUDA!(__traits(getMember, Record, m), DbProcessWith))
        {
            __traits(getMember, Record, m) =
                getUDA!(__traits(getMember, Record, m), DbProcessWith).processor(r);
        }
        else
```

skip members if some predicate says we should

# Generative serialization

```
    }
    static if(hasUDA!(__traits(getMember, Record, m), DbOnlyIf))
    {
        if(!getUDA!(__traits(getMember, Record, m), DbOnlyIf).pred(result))
            continue;
    }
    static if(hasUDA!(__traits(getMember, Record, m), DbProcessWith))
    {
        __traits(getMember, Record, m) =
            getUDA!(__traits(getMember, Record, m), DbProcessWith).processor(r))
    }
    else
    {
        alias mType = typeof(__traits(getMember, Record, m));
        static if(hasUDA!(__traits(getMember, Record, m), DbType))
            alias TypeToRead =
                getUDA!(__traits(getMember, Record, m), DbType).Type;
```

custom function to proceess the data from the row

# Generative serialization

```
        if(!getUDA!(__traits(getMember, Record, m), DbOnlyIf).pred(result))
            continue;
    }
    static if(hasUDA!(__traits(getMember, Record, m), DbProcessWith))
    {
        __traits(getMember, Record, m) =
            getUDA!(__traits(getMember, Record, m), DbProcessWith).processor(r))
    }
    else
    {
        alias mType = typeof(__traits(getMember, Record, m));
        static if(hasUDA!(__traits(getMember, Record, m), DbType))
            alias TypeToRead =
                getUDA!(__traits(getMember, Record, m), DbType).Type;
        else
            alias TypeToRead = mType;
        static if(hasUDA!(__traits(getMember, Record, m), DbName))
```

Otherwise, we are going to read it normally, but...

# Generative serialization

```
{
    __traits(getMember, Record, m) =
        getUDA!(__traits(getMember, Record, m), DbProcessWith).processor(r))
}
else
{
    alias mType = typeof(__traits(getMember, Record, m));
    static if(hasUDA!(__traits(getMember, Record, m), DbType))
        alias TypeToRead =
            getUDA!(__traits(getMember, Record, m), DbType).Type;
    else
        alias TypeToRead = mType;
    static if(hasUDA!(__traits(getMember, Record, m), DbName))
        enum fieldName =
            getUDA!(__traits(getMember, Record, m), DbName).name;
    else
        enum fieldName = m;
```

Use the correct type for reading

# Generative serialization

```
        {
            alias mType = typeof(__traits(getMember, Record, m));
            static if(hasUDA!(__traits(getMember, Record, m), DbType))
                alias TypeToRead =
                    getUDA!(__traits(getMember, Record, m), DbType).Type;
            else
                alias TypeToRead = mType;
            static if(hasUDA!(__traits(getMember, Record, m), DbName))
                enum fieldName =
                    getUDA!(__traits(getMember, Record, m), DbName).name;
            else
                enum fieldName = m;
            __traits(getMember, Record, m) = cast(mType)
                r.getWithType!TypeToRead(fieldName);
        }
    }

    return result;
```

Check for an override of the name in the DB

# Generative serialization

```
            static if(hasUDA!(__traits(getMember, Record, m), DbType))
                alias TypeToRead =
                    getUDA!(__traits(getMember, Record, m), DbType).Type;
            else
                alias TypeToRead = mType;
            static if(hasUDA!(__traits(getMember, Record, m), DbName))
                enum fieldName =
                    getUDA!(__traits(getMember, Record, m), DbName).name;
            else
                enum fieldName = m;
            __traits(getMember, Record, m) = cast(mType)
                r.getWithType!TypeToRead(fieldName);
        }
    }

    return result;
}
```

Do the actual read

# Generative serialization

```
            static if(hasUDA!(__traits(getMember, Record, m), DbType))
                alias TypeToRead =
                    getUDA!(__traits(getMember, Record, m), DbType).Type;
            else
                alias TypeToRead = mType;
            static if(hasUDA!(__traits(getMember, Record, m), DbName))
                enum fieldName =
                    getUDA!(__traits(getMember, Record, m), DbName).name;
            else
                enum fieldName = m;
            __traits(getMember, Record, m) = cast(mType)
                r.getWithType!TypeToRead(fieldName);
        }
    }

    return result;
}
```

After the loop, return the result

# GENERATIVE VS. MANUAL

- 41 lines (Generative) vs. 21 (Manual)
- Generative version DRYer than Manual
- Generative version does not need updating when struct updates
- Generative tends to compile if correct, manual bugs are more subtle.

# MIXINS AND CTFE

- String manupulation in D is strong, and mostly available for CTFE
- You truly can generate code!
- More difficult to follow
- More difficult to debug
- Allows for modular usage

# OTHER LANGUAGES

# JAVA

- Statically compiled
- Supports generics, but not templates
- Strong Runtime Type Info

```
public interface Serializable {}
```

- Java uses reflection for introspection
- Serializable interface is just a tag
- There are some hooks, but not statically defined.
- Optimization must rely on JIT

# C/C++

- Generative template capabilities
- But not as useful introspection capabilities!
- No good way to loop through members

# TRADITIONAL SERIALIZATION

```
class Archiveable {
    int x;
    template<class Archive>
    void save(Archive & ar, const unsigned int version) const {
        ar  & x;
    }
    template<class Archive>
    void load(Archive & ar, const unsigned int version) {
        ar  & x;
    }
};
```

# TRADITIONAL SERIALIZATION

```
class Archiveable {
    int x;
    template<class Archive>
    void save(Archive & ar, const unsigned int version) const
        ar  & x;
    }
    template<class Archive>
    void load(Archive & ar, const unsigned int version) {
        ar  & x;
    }
};
```

- Abstracts what the archiver does
- Focused more on supporting multiple formats than eliminating boilerplate
- You still define how to serialize all the members

# WHAT ABOUT C?

- No introspection capabilities, but has preprocessor!
- Most projects focused on serialization format, not ease of use

# IOPIPE AND JSON SERIALIZATION

# Usage for tokenizer

```
import iopipe.json.parser;

void main()
{
    auto message = `{"id" : 1, "name": "Steve"}`;
    auto parser = message.jsonTokenizer!(false);
    auto jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.ObjectStart);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.String);
    assert(jsonItem.data(parser.chain) == "id");
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Colon);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Number);
```

# Usage for tokenizer

```d
import iopipe.json.parser;

void main()
{
    auto message = `{"id" : 1, "name": "Steve"}`;
    auto parser = message.jsonTokenizer!(false);
    auto jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.ObjectStart);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.String);
    assert(jsonItem.data(parser.chain) == "id");
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Colon);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Number);
```

Declare a message (input iopipe)

# Usage for tokenizer

```
import iopipe.json.parser;

void main()
{
    auto message = `{"id" : 1, "name": "Steve"}`;
    auto parser = message.jsonTokenizer!(false);
    auto jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.ObjectStart);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.String);
    assert(jsonItem.data(parser.chain) == "id");
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Colon);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Number);
```

Create a tokenizer for that iopipe

# Usage for tokenizer

```d
import iopipe.json.parser;

void main()
{
    auto message = `{"id" : 1, "name": "Steve"}`;
    auto parser = message.jsonTokenizer!(false);
    auto jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.ObjectStart);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.String);
    assert(jsonItem.data(parser.chain) == "id");
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Colon);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Number);
```

The object starts!

# Usage for tokenizer

```
void main()
{
    auto message = `{"id" : 1, "name": "Steve"}`;
    auto parser = message.jsonTokenizer!(false);
    auto jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.ObjectStart);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.String);
    assert(jsonItem.data(parser.chain) == "id");
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Colon);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Number);
    assert(jsonItem.data(parser.chain) == "1");
```

First member and colon

# Usage for tokenizer

```d
void main()
{
    auto message = `{"id" : 1, "name": "Steve"}`;
    auto parser = message.jsonTokenizer!(false);
    auto jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.ObjectStart);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.String);
    assert(jsonItem.data(parser.chain) == "id");
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Colon);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Number);
    assert(jsonItem.data(parser.chain) == "1");
    // and so on...
}
```

The first member's value

# Usage for tokenizer

```
{
    auto message = `{"id" : 1, "name": "Steve"}`;
    auto parser = message.jsonTokenizer!(false);
    auto jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.ObjectStart);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.String);
    assert(jsonItem.data(parser.chain) == "id");
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Colon);
    jsonItem = parser.next;
    assert(jsonItem.token == JSONToken.Number);
    assert(jsonItem.data(parser.chain) == "1");
    // and so on...
}
```

And we can continue parsing data

# DESERIALIZATION

- Use template overloading to select how to deserialize
- **Primitive types**: easy, just use `to`
- **Arrays**: loop and recurse
- **Objects**: loop and recurse, using D introspection techniques!

```
import std.range.primitives;

import std.traits;
import std.typecons : Nullable;

// define some UDAs to affect serialization
struct SerializeAs {}
struct Ignore {}
struct Optional {}

SerializeAs serializeAs() { return SerializeAs.init; }
Ignore ignore() { return Ignore.init; }
Optional optional() { return Optional.init; }

void jsonExpect(ref JSONItem item, JSONToken expectedToken, string msg, string file = __FILE__, s
{
    if(item.token != expectedToken)
    {
```

Some UDA definitions for specialized behavior

```
    import std.meta;
    enum WithoutIgnore(string s) =
        !hasUDA!(__traits(getMember, T, s), Ignore());
    enum SerializableMembers =
        Filter!(WithoutIgnore, FieldNameTuple!T);
}

private void
deserializeImpl(T, JT)(ref JT tokenizer, ref T item, ReleasePolicy relPol)
    if (is(T == struct) && __traits(hasMember, T, "fromJSON"))
{
    item.fromJSON(tokenizer, relPol);
}

private void
deserializeImpl(T, JT)(ref JT tokenizer, ref T item, ReleasePolicy relPol)
    if (is(T == struct)
        && !isInstanceOf!(JSONValue, T)
```

Handle "Roll-your-own"

```
deserializeImpl(T, JT)(ref JT tokenizer, ref T item, ReleasePolicy relPol)
    if (is(T == struct) && __traits(hasMember, T, "fromJSON"))
{
    item.fromJSON(tokenizer, relPol);
}


private void
deserializeImpl(T, JT)(ref JT tokenizer, ref T item, ReleasePolicy relPol)
    if (is(T == struct)
        && !isInstanceOf!(JSONValue, T)
        && !isInstanceOf!(Nullable, T)
        && !__traits(hasMember, T, "fromJSON"))
{
    // check to see if any member is defined as the representation
    import std.traits;
    alias representers = getSymbolsByUDA!(T, SerializeAs());
    static if(representers.length > 0) {
        static assert(representers.length == 1, "Only one field can be used to represent an objec
```

Function to handle standard structs

```
    if (is(T == struct)
        && !isInstanceOf!(JSONValue, T)
        && !isInstanceOf!(Nullable, T)
        && !__traits(hasMember, T, "fromJSON"))
{
    // check to see if any member is defined as the representation
    import std.traits;
    alias representers = getSymbolsByUDA!(T, SerializeAs());
    static if(representers.length > 0) {
        static assert(representers.length == 1, "Only one field can be used to re
        deserializeImpl(tokenizer,
                        __traits(getMember, item,
                                 __traits(identifier, representers[0])), relPol);
    }
    else
    {
        // expect an object. We want to deserialize the JSON data
        alias members = SerializableMembers!T;
```

Check for structs that are parsed into one member

```
    alias representers = getSymbolsByUDA!(T, SerializeAs());
    static if(representers.length > 0) {
        static assert(representers.length == 1, "Only one field can be used to represent an obje
        deserializeImpl(tokenizer,
                        __traits(getMember, item,
                                 __traits(identifier, representers[0])), relPol);
    }
    else
    {
        // expect an object. We want to deserialize the JSON data
        alias members = SerializableMembers!T;
        bool[members.length] visited;

        // any members that are optional, mark as already visited
        static foreach(idx, m; members) {
            static if(hasUDA!(__traits(getMember, T, m), Optional()))
                visited[idx] = true;
        }
```

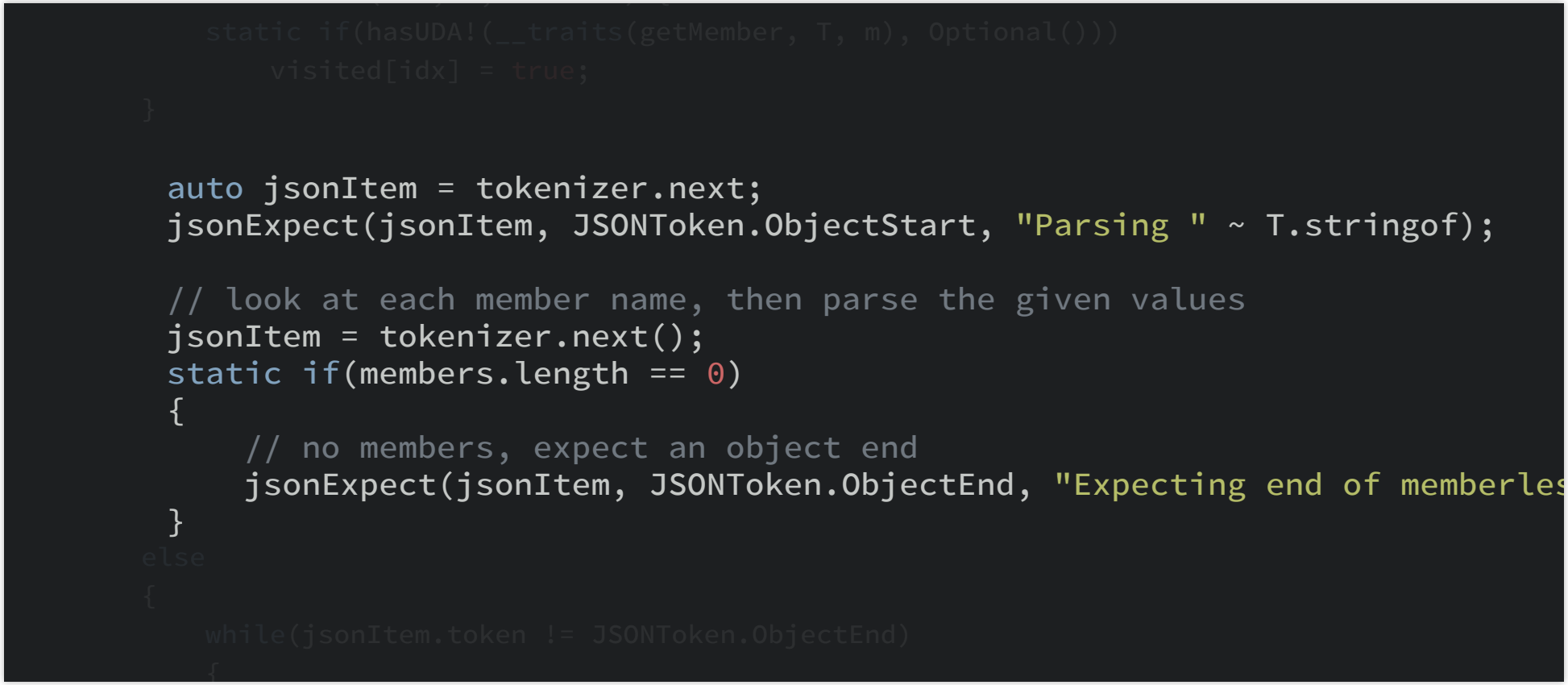
Otherwise, get the list of members we will serialize

```
    item = jsonItem.data(tokenizer.chain).to!T;
}

// TODO: need to ignore function members
// TODO: need to use UDAs to drive how this works
private template SerializableMembers(T)
{
    import std.traits;
    import std.meta;
    enum WithoutIgnore(string s) =
        !hasUDA!(__traits(getMember, T, s), Ignore());
    enum SerializableMembers =
        Filter!(WithoutIgnore, FieldNameTuple!T);
}

private void
deserializeImpl(T, JT)(ref JT tokenizer, ref T item, ReleasePolicy relPol)
    if (is(T == struct) && __traits(hasMember, T, "fromJSON"))
```

Filter out ignored members

```
    bool[members.length] visited;

    // any members that are optional, mark as already visited
    static foreach(idx, m; members) {
        static if(hasUDA!(__traits(getMember, T, m), Optional()))
            visited[idx] = true;
    }
```

Assume optionals are visited already




Special case no members

```
        // no members, expect an object end
        jsonExpect(jsonItem, JSONToken.ObjectEnd, "Expecting end of memberless object " ~ T.
    }
    else
    {
        while(jsonItem.token != JSONToken.ObjectEnd)
        {
            if(jsonItem.token == JSONToken.Comma)
                jsonItem = tokenizer.next();

            jsonExpect(jsonItem, JSONToken.String, "Expecting member name of
            auto name = jsonItem.data(tokenizer.chain);

            jsonItem = tokenizer.next();
            jsonExpect(jsonItem, JSONToken.Colon, "Expecting colon when pars
OBJ_MEMBER_SWITCH:
            switch(name)
            {
```

Standard parsing fare

```
OBJ_MEMBER_SWITCH:
                switch(name)
                {
                    static foreach(i, m; members)
                    {
                    case m:
                        tokenizer.deserializeImpl(__traits(getMember, item, m),
                                                  relPol);
                        visited[i] = true;
                        break OBJ_MEMBER_SWITCH;
                    }

                default:
                    import std.format : format;
                    throw new Exception(format("No member named '%s' in type `%s`
                }
```

The generative part


Verify all non optional members were found

# VIBE.D

# EXPOSING A D API ON THE WEB

- Use all available information
- Map names intuitively
- Where custom behavior is needed, use UDAs

# Encapsulate the API in a class

```
import vibe.vibe;
import vibe.web.auth;

@path("/wh")
class WebHandler
{
    string[string] items;

    void getIndex(HTTPServerResponse res)
    {
        auto items = this.items;
        res.render!("index.dt", items);
    }

    void postItem(HTTPServerRequest req, string name, string value,
```

# Encapsulate the API in a class

```
import vibe.vibe;
import vibe.web.auth;

@path("/wh")
class WebHandler
{
    string[string] items;

    void getIndex(HTTPServerResponse res)
    {
        auto items = this.items;
        res.render!("index.dt", items);
    }

    void postItem(HTTPServerRequest req, string name, string value,
```

Declare the class to handle route '/wh'

# Encapsulate the API in a class

```
import vibe.vibe;
import vibe.web.auth;

@path("/wh")
class WebHandler
{
    string[string] items;

    void getIndex(HTTPServerResponse res)
    {
        auto items = this.items;
        res.render!("index.dt", items);
    }

    void postItem(HTTPServerRequest req, string name, string value,
```

Data storage for the example

# Encapsulate the API in a class

```
@path("/wh")
class WebHandler
{
    string[string] items;

    void getIndex(HTTPServerResponse res)
    {
        auto items = this.items;
        res.render!("index.dt", items);
    }

    void postItem(HTTPServerRequest req, string name, string value,
                  HTTPServerResponse res)
    {
        items[name] = value;
```

Simple get route

# Encapsulate the API in a class

```
{
    auto items = this.items;
    res.render!("index.dt", items);
}

void postItem(HTTPServerRequest req, string name, string value,
              HTTPServerResponse res)
{
    items[name] = value;
    res.redirect("/wh/index");

}

@path("item/:name")
void getItem(string _name, HTTPServerResponse res)
{
```

Store an item

# Encapsulate the API in a class

```
        items[name] = value;
        res.redirect("/wh/index");

    }


    @path("item/:name")
    void getItem(string _name, HTTPServerResponse res)
    {
        if(auto v = _name in items)
            res.writeBody("<html><body>Item named " ~ _name ~ " is "
                          ~ *v ~ "<br><a href='/wh/index'>Home</a></body></html>",
                          "text/html");
        else
            res.redirect("/wh/index");
    }
```

Retrieve an item

# Add authentication

```
import vibe.vibe;
import vibe.web.auth;

struct AuthInfo
{
    string username;
@safe:
    bool isEditor() {
        return username == "root";
    }

    bool isViewer(string _name) {
        return _name.length > 0 && _name[0] == 'a';
    }
}
```

# Add authentication

```
import vibe.web.auth;

struct AuthInfo
{
    string username;
@safe:
    bool isEditor() {
        return username == "root";
    }

    bool isViewer(string _name) {
        return _name.length > 0 && _name[0] == 'a';
    }
}

@path("/web2")
```

The authorization state

# Add authentication

```
    bool isViewer(string _name) {
        return _name.length > 0 && _name[0] == 'a';
    }
}

@path("/wh2")
@requiresAuth
class WebHandler2

{
    string[string] items;

    @noAuth
    void getIndex(HTTPServerResponse res)
    {
```

Now, our class requires authorization

# Add authentication

```
                    ~ *v ~ "<br><a href='/wh2/index'>Home</a></body></html>",

                    "text/html");
    else
        res.redirect("/wh2/index");
}

@safe @noRoute AuthInfo
    authenticate(scope HTTPServerRequest req,
                 scope HTTPServerResponse res)
{
    // terrible authentication mechanism
    AuthInfo result;
    result.username = req.form.get("username", "anonymous");
    return result;
}
```

And the class defines how to get authentication

# Add authentication

```
@path("/wh2")
@requiresAuth
class WebHandler2

{
    string[string] items;

    @noAuth
    void getIndex(HTTPServerResponse res)
    {
        auto items = this.items;
        res.render!("index.dt", items);
    }

    @auth(Role.editor)
```

Listing the items doesn't require any auth

# Add authentication

```
@noAuth
void getIndex(HTTPServerResponse res)
{
    auto items = this.items;
    res.render!("index.dt", items);
}

@auth(Role.editor)
void postItem(HTTPServerRequest req, string name, string value,
              HTTPServerResponse res)
{
    items[name] = value;
    res.redirect("/wh2/index");
}
```

But setting an item does

# Add authentication

```
void postItem(HTTPServerRequest req, string name, string value,
              HTTPServerResponse res)
{
    items[name] = value;
    res.redirect("/wh2/index");
}

@auth(Role.viewer)
@path("item/:name")
void getItem(string _name, HTTPServerResponse res)
{
    if(auto v = _name in items)
        res.writeBody("<html><body>Item named " ~ _name ~ " is "
                      ~ *v ~ "<br><a href='/wh2/index'>Home</a></body></html>",
```

`_name` is passed to `authinfo.isViewer`

# DEMO OF VIBE.D

# FIXING THE DRAWBACKS

# COMPILE TIME IS IMPORTANT

- template and generative programming are compile-time killers.
- vibe.d default installation takes 4.5 seconds to compile on my 2014 Mac.
- CTFE and templates are suspects in compile time bloaters.

# AVOID THE BOILERPLATE

- Any time you can avoid boilerplate, you should.
- Have the compiler write your boilerplate, it's better at it.
- Generative programming is the crown jewel of D!

# YOU HAVE A D COMPILER, USE IT!

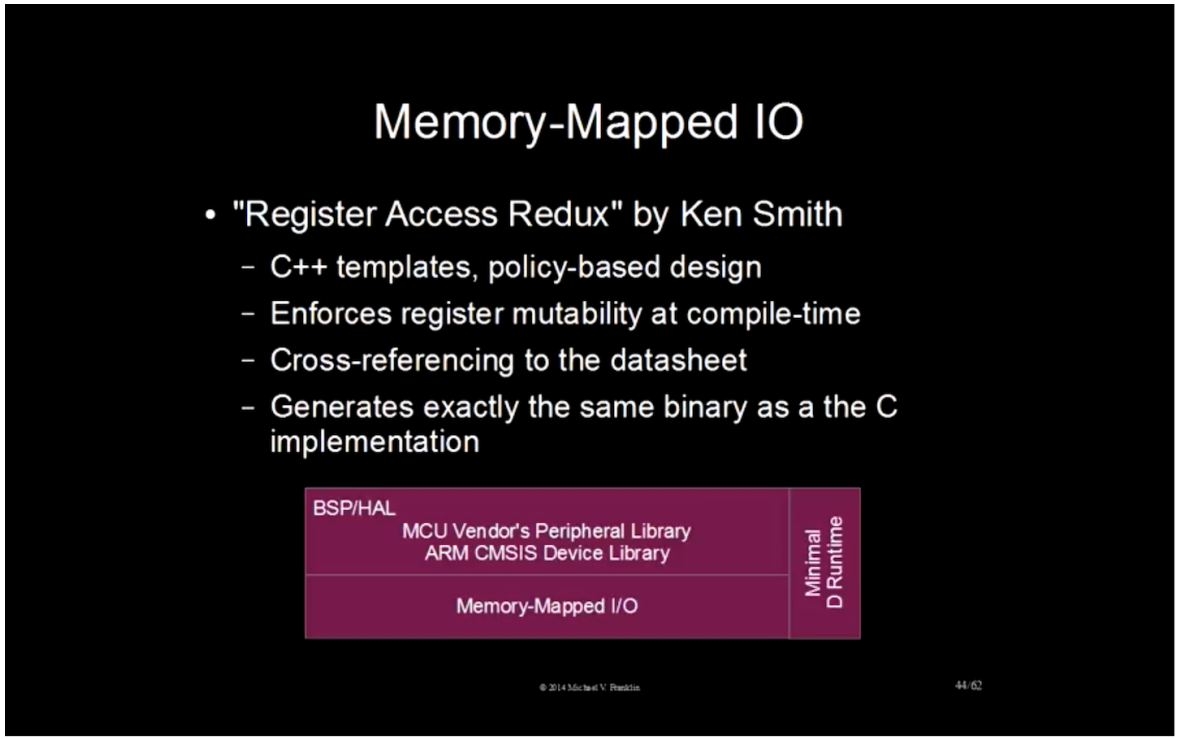
# D POWERING DEVICES

# WHAT IF D COULD RUN ON DRONES?

- Dconf 2014 - Michael V. Franklin "Tiny Ubiquitous Machines Powered by D"
- Used spec sheet text directly to create register maps
- Well, almost directly

# USE TEMPLATES TO GENERATE PERFECT BOILER PLATE REGISTER MAPS

*library: https://github.com/JinShil/memory_mapped_io*



# USE TEMPLATES TO GENERATE PERFECT BOILER PLATE REGISTER MAPS

- Enforces all reads and writes are done with as few instructions as possible, and with the correct access type.
- Generated access was 6.5x faster than the chip supplier's C library call
- Once you map the spec to code, ALL the boilerplate is solved for every register
- 1700 page spec

# CHIP SPEC CAN BE USED AS CODE?

- Imagine if you can simply import the spec, and process it.
- The spec becomes the code, transpiled by the compiler at compile time to the most efficient register mapping possible
- No more discrepencies between code libraries and spec!