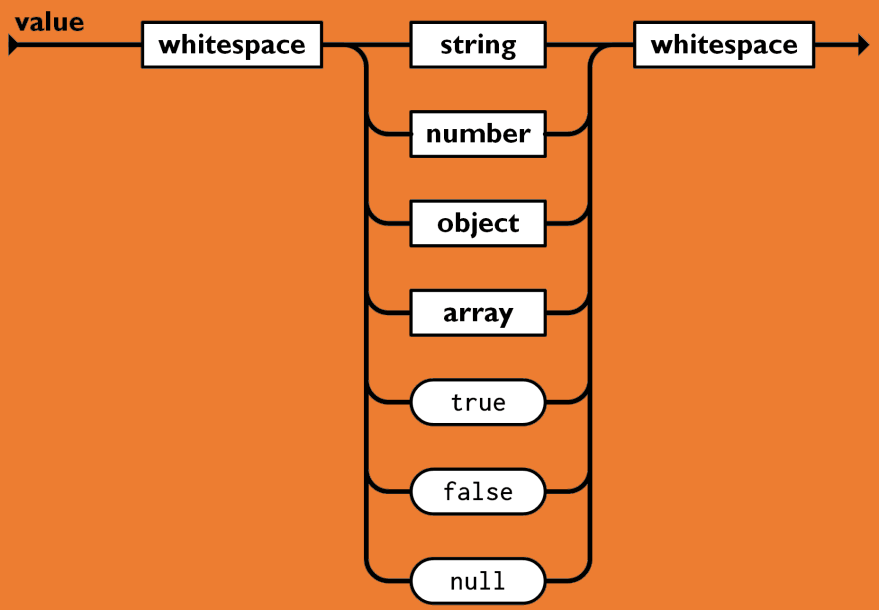
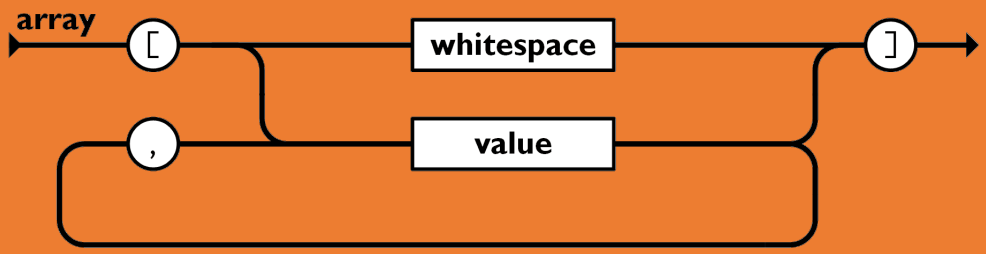
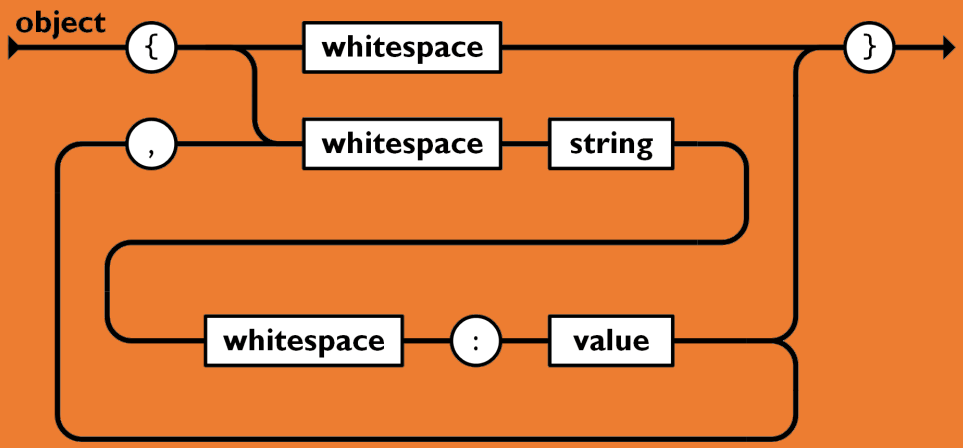
Generate JSON with DATA-GEN in RPG



by
Scott Klement





What?

This session assumes you want to create a JSON document in RPG.

For example:

```
{
  "name": "Scott Klement",
  "street": "8825 S Howell Avenue Ste 301",
  "city": "Oak Creek",
  "state": "WI",
  "postal": "53154"
}
```

How?

It's easy with the DATA-GEN opcode.

```
dcl-s Json varchar(1000);

dcl-ds address qualified;
  name   varchar(30)  inz('Scott Klement');
  street varchar(30)  inz('8825 S Howell Avenue Ste 301');
  city   varchar(20)  inz('Oak Creek');
  state  char(2)      inz('WI');
  postal varchar(10)  inz('53154');
end-ds;

DATA-GEN address %DATA(Json) %GEN('YAJLDTAGEN');
```

Yeah. It's easy. DATA-GEN put the document in the Json variable.

Why?

Each JSON thing has an RPG equivalent.

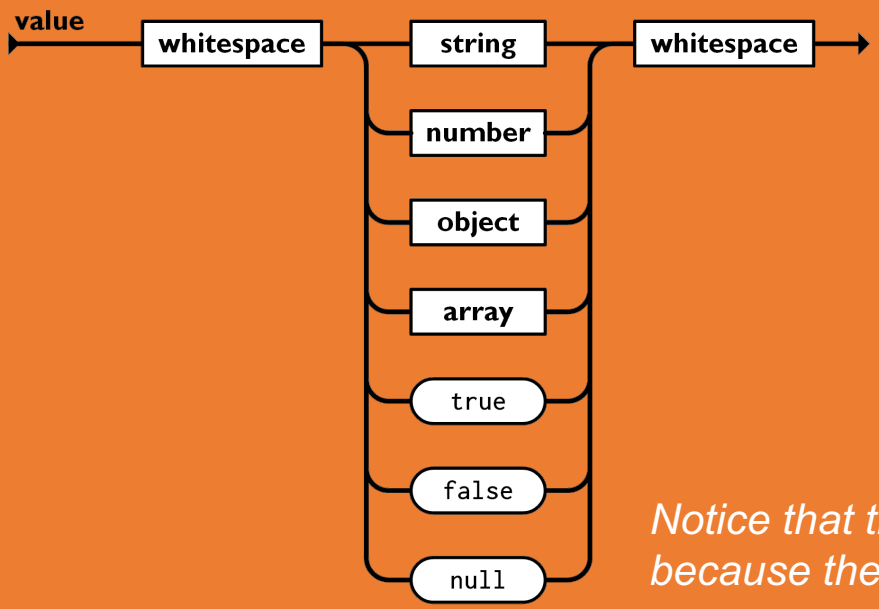
DATA-GEN makes the JSON thing from the RPG thing.

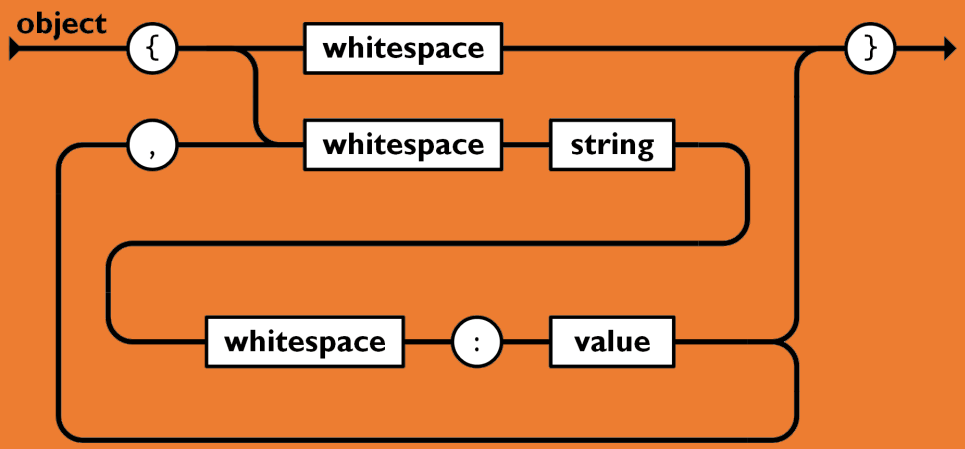
```
{
  "sub field 1": 123.45,
  "sub field 2": "string goes here",
  "accepted": true,
  "days open": [ "Monday", "Wednesday", "Friday" ]
}
```

Characters	Json Meaning	RPG Equivalent
"string goes here"	Character string	CHAR or VARCHAR
123.45	Number	Packed, Zoned, Int, Float, etc
true	Boolean (true or false)	Indicator (*ON or *OFF)
{ "field": "value" }	Object	Data Structure
[1, 2, 3]	Array	DIM

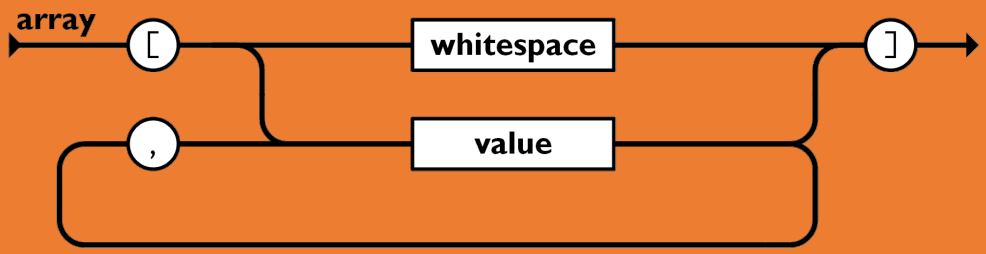
That title slide, again.

All of my examples so far have been objects.





Notice that they refer to each other because they can be nested.

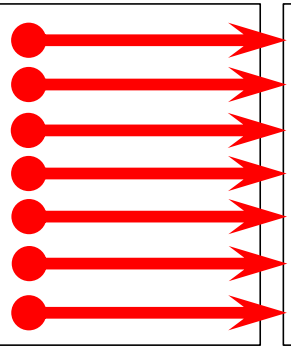


Look Again

Makes JSON things from RPG things.

```
DATA-GEN address %DATA(Json) %GEN('YAJLDTAGEN');
```

```
dcl-ds address qualified;
  name    varchar(30);
  street  varchar(30);
  city    varchar(20);
  state   char(2);
  postal  varchar(10);
end-ds;
```



```
{
  "name": "string",
  "street": "string",
  "city": "string",
  "state": "string",
  "postal": "string"
}
```

Characters	Json Meaning	RPG Equivalent
"string goes here"	Character string	CHAR or VARCHAR
123.45	Number	Packed, Zoned, Int, Float, etc
true	Boolean (true or false)	Indicator (*ON or *OFF)
{ "field": "value" }	Object	Data Structure
[1, 2, 3]	Array	DIM

What the parts of DATA-GEN mean and do.

More Details

```
DATA-GEN source-variable %DATA(result {: options}) %GEN(generator {: options});
```

- source-variable: RPG variable (usually a data structure) to generate the structured document from.
- result: Specifies the result variable, either as a character variable (default) or as an IFS pathname to write to.
- result options: Space-separated list of options that control how RPG transfers data from your source variable into the result (more to come!)
- generator: Third-party program or service program that will generate the document. The generator is what determines the format of the document you're generating.
- generator options: Character literal or RPG variable that contains options used by the generator. The format of this variable is defined by the generator program and will be different for each generator you use.

Other Options

Many options exist for %DATA.

Here are the most commonly used ones.

- doc – controls where the document is generated string (default) or file.
- countprefix – control the number of specified elements generated
- renameprefix – lets you specify variables containing alternate names for subfields.

```
%DATA(myStmf:'put options here')
```

Writing a File

doc=file changes the first parameter to %DATA to be an IFS path name, then writes there.

```
dcl-s MyFile varchar(100);

MyFile = '/home/scott/address.json';
data-gen address %data(MyFile: 'doc=file') %gen('YAJLDTAGEN');
```

Variable Length Arrays

How to deal with variable-length arrays? Example: An invoice has a variable number of items on it.

```
{
  "items": [
    { "itemNo": 1001, "desc": "Some Description",   "qty":  12, "price":  51.99 },
    { "itemNo": 1002, "desc": "Second Description", "qty":   6, "price":  94.10 },
    { "itemNo": 1003, "desc": "Third Description",  "qty":  20, "price":  12.00 },
    { "itemNo": 1004, "desc": "Silly Things",       "qty": 104, "price":   3.75 },
    { "itemNo": 1005, "desc": "Some other things",  "qty":   3, "price": 101.06 }
  ]
}
```

```
dcl-ds invoice qualified;
  dcl-ds items dim(999);
    itemNo packed(5: 0);
    desc   varchar(30);
    qty    packed(5: 0);
    price  packed(7: 2);
  end-ds;
end-ds;
```

A DS like this would be a problem. It would output 999 elements.

CountPrefix

How to deal with variable-length arrays? Example: An invoice has a variable number of items on it.

```
{
  "items": [
    { "itemNo": 1001, "desc": "Some Description",   "qty":  12, "price":  51.99 },
    { "itemNo": 1002, "desc": "Second Description", "qty":   6, "price":  94.10 },
    { "itemNo": 1003, "desc": "Third Description",  "qty":  20, "price":  12.00 },
    { "itemNo": 1004, "desc": "Silly Things",       "qty": 104, "price":   3.75 },
    { "itemNo": 1005, "desc": "Some other things",  "qty":   3, "price": 101.06 }
  ]
}
```

```
dcl-ds invoice qualified;
  num_items int(10);
  dcl-ds items dim(999);
    itemNo packed(5: 0);
    desc   varchar(30);
    qty    packed(5: 0);
    price  packed(7: 2);
  end-ds;
end-ds;

MyFile = '/home/scott/invoice.json';
data-gen invoice %data(MyFile: 'doc=file countprefix=num ') %gen('YAJLDTAGEN');
```

Any DS subfield prefixed by num_ will be the count of another element.

Special Names

When a name in a JSON document isn't a possible RPG field name...

```
{
  "customer name": "string",
  "street address": "string",
  "city": "string",
  "state": "string",
  "postal": "string"
}
```

This will NOT work; RPG doesn't allow spaces or quotes in a variable name:

```
dcl-ds address qualified;
  "customer name"   varchar(30);
  "street address"  varchar(30);
   city             varchar(20);
   state            char(2);
   postal           varchar(10);
end-ds;
```

RenamePrefix

When a name in a JSON document isn't a possible RPG field name...

```
dcl-ds address qualified;
  name          varchar(13);
  name_name     varchar(30) inz('customer name');
  street        varchar(30);
  name_street   varchar(14) inz('street address');
  city          varchar(20);
  state         char(2);
  postal   varchar(10);
end-ds;

data-gen invoice %data(MyFile: 'doc=file renameprefix=name_')
                 %gen('YAJLDTAGEN');
```

```
{
  "customer name": "string",
  "street address": "string",
  "city": "string",
  "state": "string",
  "postal": "string"
}
```

Anything prefixed by name_ controls the output name of the variable.

More Options

- There are other configuration options you can use -- these were the most common ones.
- JSON is not the only option. Can do others, XML, YAML, HTML, more.
- YAJLDTAGEN is not the only option.
- You can even write your own.

The End.

- It's simple.
- But can be less simple (with lots of options) when needed.
- Assuming you need to generate a JSON document -- DATA-GEN lets you do that.
- Thank you.