C3.ai Academy

C3.ai Development Core Quick Reference Guide

Table of Contents

How to use this document:

This is a work aid for **work**. It covers skills you build in class and is useful as a new developer on you projects at work. The LightBulb diagrams are specific to the C3.ai Development Core course but are a good refence for your own ERDs.

If you can't find the answer here, then go to the C3 AI Developer Portal:
https://developer.c3.ai/

And look at Documentation:
https://developer.c3.ai/documentation

If you need more help, then try C3 AI Community:
https://community.c3.ai/

And if you need even more help, talk the developers and C3.ai employees on your COE team.

Naming Conventions

- **Type names**
  - **PascalCase** (first letter upper case)
  - Very descriptive with no abbreviations

- **Field names**
  - **camelCase** (first letter lower case)
  - Very descriptive with no abbreviations
  - Use plurals for collection fields

- **Schema names** and **type keys**
  - **UPPR_SNK_CS** or upper snake case - all caps separated by an underscore
  - Use abbreviations by removing all non-leading vowels.
    For example:
    - Upper snake case becomes UPPR_SNK_CS
    - SmartBulb Measurements becomes SMRTBLB_MSRMNTS

  - Schema labels have a 30 character limit - so keep them as short as possible
    - Best Practice is < 15 characters

Console Commands

```
// Run this command after you provision
// EVERY TIME
c3ImportAll()
```

Documentation - Console Commands

```
// View Type documentation
c3ShowType(MyType)

// View Type documentation in a new help tab
c3Doc(MyType)

// View documentation for Fields
c3ShowType(ValueType)

// View documentation for the constraint annotation.
// You can use this syntax for Db and Ts annotations.
c3ShowType(Ann.Constraint)

// Search documentation in a new help tab
c3Doc("My Search String")
```

Fetch and Evaluate Method - Console Commands

```
// Fetch data from a persistable type as JSON in the dev console
MyType.fetch()

// View data from a persistable type in a table in static console
c3Grid(MyType.fetch())

// Fetch and return a collection field
MyType.fetch({include: 'this, myCollections'})

// fetch method with a filter function
c3Grid(MyType.fetch({filter: "myField =='myString'"}))

// Evaluate method wrapped in c3Grid
c3Grid(MyType.evaluate({projection: 'id', filter: "myField == 'myString'"}))

// Evaluate method with count method
MyType.evaluate({projection: "myField.id", filter: "id == 'myString'"}).count

// View documentation for Fetch Specifications
c3ShowType(FetchSpec)

// For more about the Evaluate Specifications
c3ShowType(EvaluateSpec)
```

Create and Remove Type Instances - Console Commands

```
// Create an instance of a Type - id auto-generated
MyType.create({myRequiredField: 'myString'})

// Create an instance of a Type - id and a numerical field specified
MyType.create({id: "MYID1", myRequiredField: 100})

// Create many instances of a Type
MyType.createBatch([
    {id: "MYID2", myRequiredField: "myString"},
    {id: "MYID3", myRequiredField: "myString"}
])

// Remove a Type Instance
MyType.remove({id: "MYID1"})

// Remove all instances of a Type
MyType.removeAll()

// Remove all data from a Tag
Tag.get('MyTagId').removeData(true,true)
```

Merge and Upsert Methods - Console Commands

```
//Merge new data into a type instance
MyType.merge({id: "id1", myReference: "idB"})

//Upsert new data into a type instance
MyType.upsert({id: "id1", myRequiredField: "myString"})

//Merge new data into a type instance - with timedValueHistoryField
MyType.merge({id: "id1", myTimedValue: {"value": 0}})
```

Import API and Normalization - Console Commands

```
// Generate an Authentication Token
// Good only for the current browser session!
Authenticator.generateC3AuthToken()

// View Invalidation Queue
c3Grid(InvalidationQueue.countAll())

// View Status of import
c3Grid(SourceFile.get("MySourceCollection_MySourceFile.csv").sourceStatus())

// Fetch method with limit spec
c3Grid(MyType.fetch({limit:500}))

// Trigger normalization via a header Type
MyTimeSeriesHeaderType.refreshNormalization({force: true, async: true})

// View normalized time series data
var normalizedData = NormalizedTimeseriesPersister.getId(
    id = MyHeaderType.normalizedTimeseriesKey(
        headerId = "MyHeaderID",
        tsField = "MyQuantityField"
    ),
    spec = NormalizedTimeseriesGetSpec.make({
        timeRange: NormalizedTimeseriesRange.make({
            start: "2015-01-01",
            end: "2019-06-01"
        })
    })
);
c3Grid(normalizedData.values)
```

Metrics - Console Commands

Make and visualize a SimpleMetric - Console Commands

```
var s_metric = SimpleMetric.make({
    "id": "MyMetric_MyType",
    "name": "MyMetric",
    "srcType": "MyType",
    "path": "myPath.myField",
    "expression": "avg(avg(normalized.data.myDataField))"
});

var s_spec = EvalMetricsSpec.make({
  ids: ["myID1"],
  expressions: ["MyMetric"],
  start: "2015-01-01",
  end: "2019-06-01",
  interval: "DAY"
});

c3Viz(MyType.evalMetricsWithMetadata(s_spec,[s_metric]));
```

Make and visualize a TSDecl SimpleMetric - Console Commands

```
var ts_metric = SimpleMetric.make({
  "id": "MyTSDMetric_MyType",
  "name": "MyTSDMetric",
  "srcType": "MyType",
  "path": "myPath.myField",
  "tsDecl": {
    "data": "myDataTypeRef",
    "start": "myTimstampField",
    "value": "myValueField",
    "treatment": "PREVIOUS",
  }
});

var ts_spec = EvalMetricsSpec.make({
  ids: ["MyID"],
  expressions: ["MyTSDMetric"],
  start: "2015-01-01",
  end: "2019-06-01",
  interval: "DAY"
});

c3Viz(MyType.evalMetricsWithMetadata(ts_spec,[ts_metric]));
```

Make and visualize a CompoundMetric - Console Commands

```
var c_metric = CompoundMetric.make({
    "name": "MyCMetric",
    "id": "MyCMetric",
    "expression": "MyExpression"
});

var c_spec = EvalMetricsSpec.make({
  ids: ["MyID"],
  expressions: ["MyCMetric"],
  start: "2015-01-01",
  end: "2019-06-01",
  interval: "DAY"
});

c3Viz(MyType.evalMetricsWithMetadata(c_spec,[c_metric]));
```

Visualize any Metric - Console Commands

```
c3Viz(MyType.evalMetrics(EvalMetricsSpec.make({
  ids: ["MyId1", "MyId2"],
  expressions: ["MyMetricA", "MyMetricB"],
  start: "2015-01-01",
  end: "2015-02-03",
  interval: "HOUR"
})))
```

Type Definitions

MyPersistableType.c3typ

```
/**
* MyType.c3typ
* A widget unit in a {@link Gadget} with many {@link Gizmo}s in it
*/
entity type MyType schema name "M_TP" {

    // The number of gaggles in this widget - Primitive field
    gaggle: double

    // The gadget that this widget is in - Required Reference field
    gadget: !Gadget

    // A list of all the gizmos in this widget - Collection field
    gizmos: [Gizmo](widget)
}
```

SourceMyType.c3typ

```
/**
* SourceMyType.c3typ
* This type represents the raw data for {@link MyType} information.
*/
type SourceMyType mixes Source {

    // This represents the id of a gizmo
    gizmo: string
}
```

TransformSourceMyTypeToMyTargetType.c3typ

```
/**
* TransformSourceMyTypeToMyType.c3typ
* Transforms data from {@link SourceMyType} to {@link MyType}
*/
type TransformSourceMyTypeToMyType mixes MyType transforms SourceMyType {

    // The id of a MyType - Primative Field
    id: ~ expression "myTypeId"

    // A reference to the id of gadget - Reference Field
    gadget: ~ expression {id: "gadget"}

    // Expression engine concatenate function
    id: ~ expression "concat(SourcFieldA,'myString',SourceFieldB)"
}
```

MyExternalType.c3typ

```
/**
* MyExternalType.c3typ
* A blurb containing many {@link Blob}s.
*/
@db(datastore="myDataStore")
entity type MyExternalType mixes External, NoSystemCols, MetricEvaluatable
schema name "M_EXTRNL_TP" {

    // The id for a widget - the Primary Key for the external table
    id: ~ schema name 'primary_key'

    // The full name of the widget - Primitive Type
    fullName: string schema name 'widget_name'

    // List of gizmos in this widget - Collection Field
    gizmos: [Gizmo](widget)

    // The gadget this widget is in - Reference Field
    gadget: Gadget schema name 'gadget_id'
}
```

MyExtendableType.c3typ

```
/**
* MyExtendableType.c3typ
* A blurb containing many {@link Blob}s.
*/
extendable entity type MyExtendableType schema name "M_EXTNDBL_TP"{

    // The number widgets that are inside this blurb - Enumerated Values
    widget: double enum(5, 10 ,15 ,22)
}
```

MyExtendedType.c3typ

```
/**
* MyExtendedType.c3typ
* A blurb containing many {@link Blob}s.
*/
entity type MyExtendedType extends MyExtendableType type key  "M_EXTNDD_TP"{

    // The widget this blurb is in - Default Value
    widget: Widget post default "{id: 'myId'}"
}
```

MyTimedValueHistoryType.c3typ

```
/**
* MyTimedValueHistoryType.c3typ
* Sparse timestamped events for {@link MyParentType}.
*/
entity type MyTimedValueHistoryType mixes TimedValueHistory<MyParentType>,
Integer schema name "M_TMD_VL_HSTR"
```

MyTimedValueType.c3typ

```
/**
* MyTimedValueType.c3typ
* The current timestamped event value for {@link MyParentType}.
* Populated at runtime.
*/
type MyTimedValueType mixes TimedValue, Integer
```

MyTimedIntervalRelation.c3typ

```
/**
* MyTimedIntervalRelation.c3typ
* Describes a timed interval relationship between a {@link MyFromType} and a
{@link MyToType}.
*/

// These records will be indexed on two fields; the to and from fields.
@db(index=['to', 'from'])
entity type MyTimedIntervalRelation mixes TimedIntervalRelation<MyFromType,
MyToType> schema name "M_TMD_INTRVL_RLTN"
```

Field syntax for stored calc and @db order annotations

```
    // The collection of {@link MyTimedIntervalRelation}s for fromType record
    @db(order='descending(start)')
    myToTypeHistory: [MyTimedIntervalRelation](from)

    // The current {@link ToType} for this fromType record
    currentMyToType: MyToType stored calc 'toTypeHistory[0].(end == null).to'
```

MyTimedValueHeader.c3typ

```
/*
* MyTimedValueHeader.c3typ
* A series of measurements taken from a single {@link MyType}.
*/
entity type MyTimedValueHeader mixes TimedDataHeader<MyTimedValuePoint>
schema name "M_TMD_DT_HDR" {

  // The the {@link MyType} for these measurements
  myType: !MyType
}
```

MyTimedDataPoint.c3typ

```
/**
* MyTimedDataPoint.c3typ
* A single measurement taken from a single {@link MyType}
*/
@db(datastore='cassandra',
 partitionKeyField='parent',
 persistenceOrder='start',
 persistDuplicates=false,
 compactType=true)
entity type MyTimedDataPoint mixes TimedDataPoint<MyTimedValueHeader>  schema
name 'M_TMD_DT_PNT' {

 // The status of the MyType
 @ts(treatment='previous')
 myNoUnitFiled: int

 // The measured value of of MyType
 @ts(treatment='avg', unitPath="parent.myPath.myUOMField")
 myUnitField: double
}
```

@db Annotations

@db order Annotation

```
    // Order collection query results by fields listed.
    @db(order='descending(myFieldA), ascending(myFieldB)')
```

@db timedValueHistoryField Annotation

```
    // timedValueHistoryField connection to TimedValue Type
    @db(timedValueHistoryField='myTimedValueHistoryField')
    myTimedValue:  MyTimedValue
```

Seed Folder JSON Type Instances

Repository and Package JSON

repository.json

```
{
  "name": "MyUniqueRepositoryName",
  "version": "0.1.0",
  "description": "My Training Repository",
  "dependencies": { "server" : "6.6.22" }
}
```

package.json

```
{
  "name": "MyUniquePackageName",
  "description": "My package description",
  "author": "My Team Name",
  "dependencies": ["standardDependencies", "machineLearning"]
}
```

File Source Types

MyFileSourceCollection.json

```
{
  "id": "MySourceCollection",
  "name": "MySourceCollection",
  "source": {
    "typeName": "MySourceType"
  },
  "sourceSystem": {
    "id": "Legacy"
  }
}
```

Metric Types

MySimpleMetric.json

```
{
    "id": "MyMetric_MyType",
    "name": "MyMetric",
    "srcType": "MyType",
    "path": "myPath.myField",
    "expression": "avg(avg(normalized.data.myDataField))"
}
```

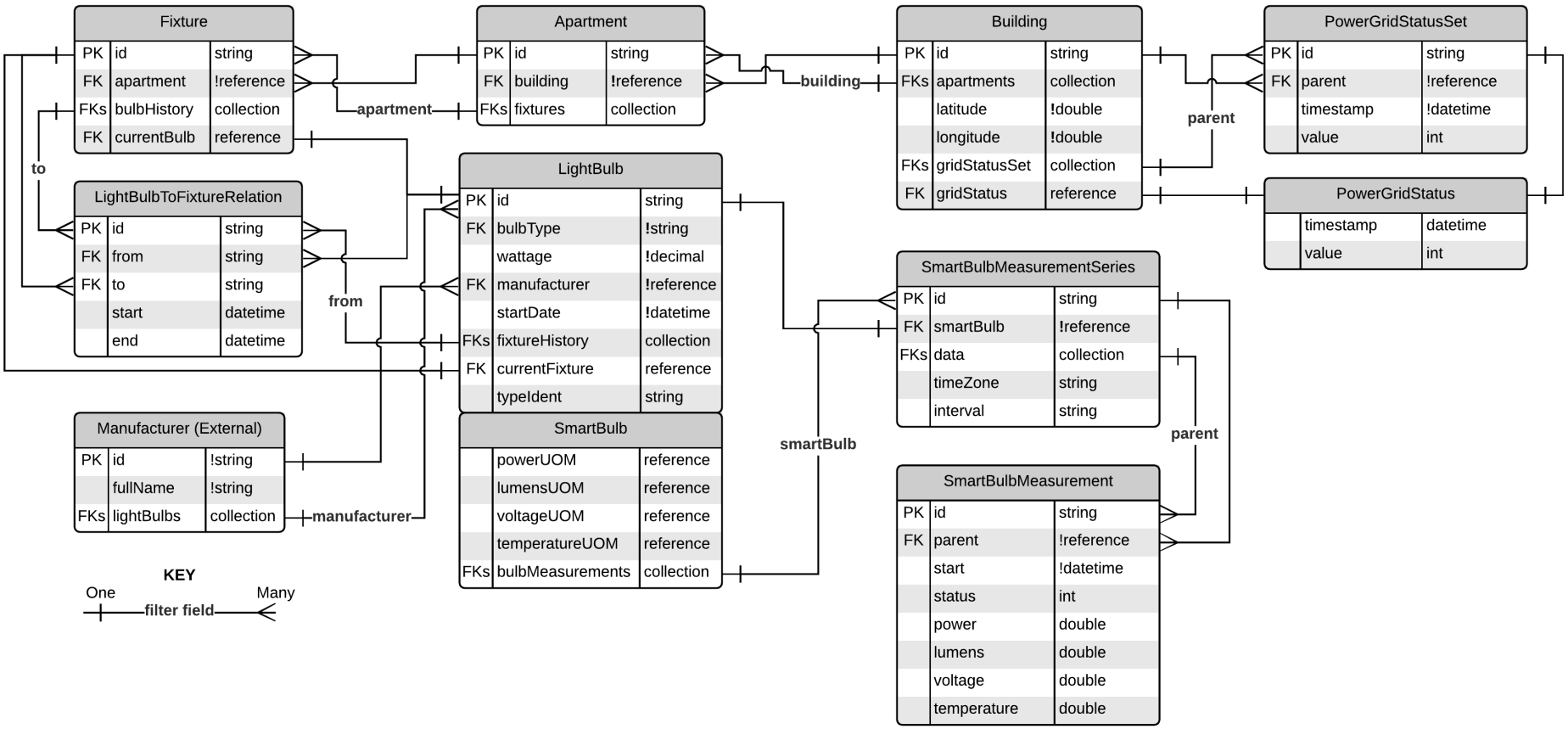
MyTSDeclSimpleMetric.json

```
{
  "id": "MyTSDMetric_MyType",
  "name": "MyTSDMetric",
  "srcType": "MyType",
  "path": "myPath.myField",
  "tsDecl": {
    "data": "myDataTypeRef",
    "start": "myTimstampField",
    "value": "myValueField",
    "treatment": "PREVIOUS",
  }
}
```

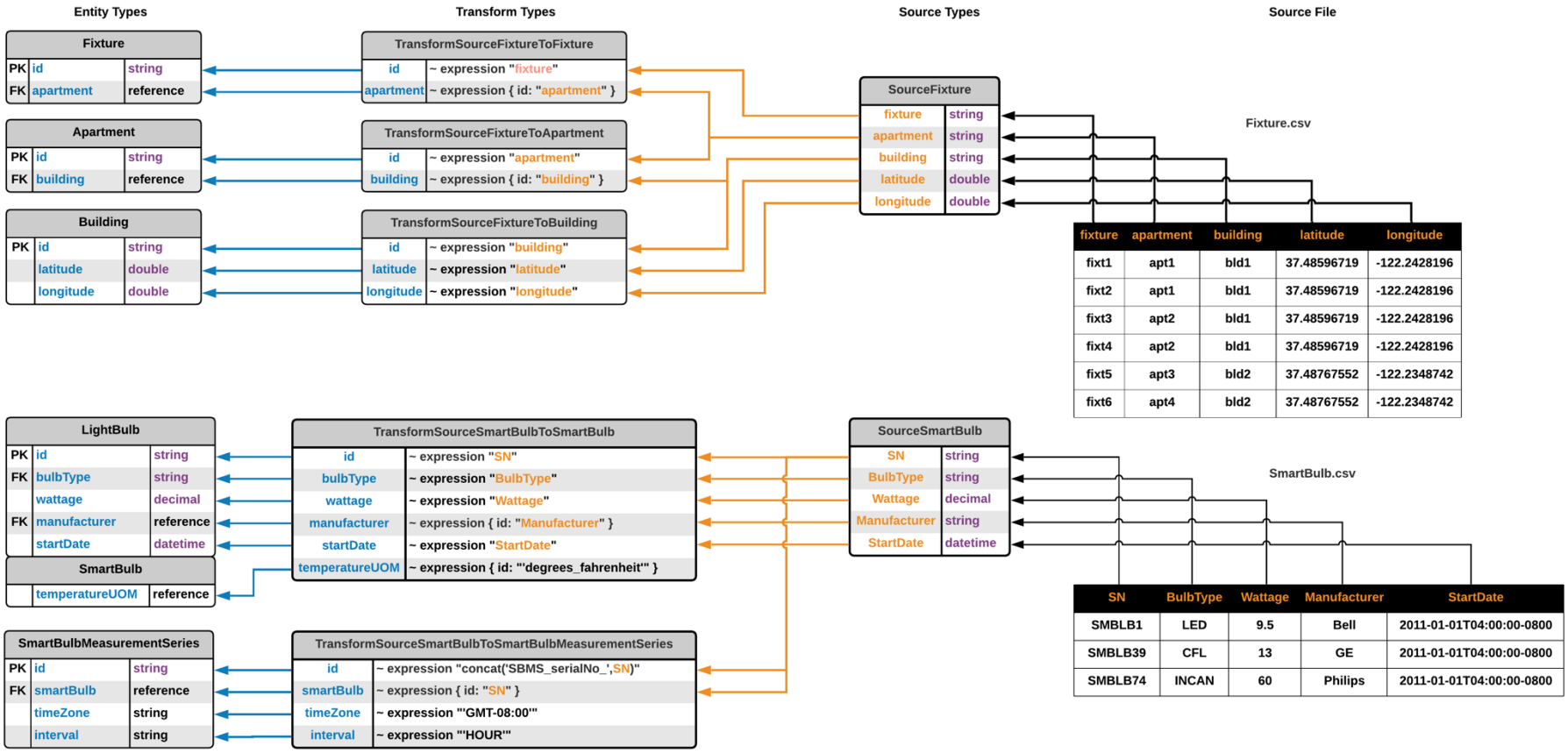
MyCompoundMetric.json

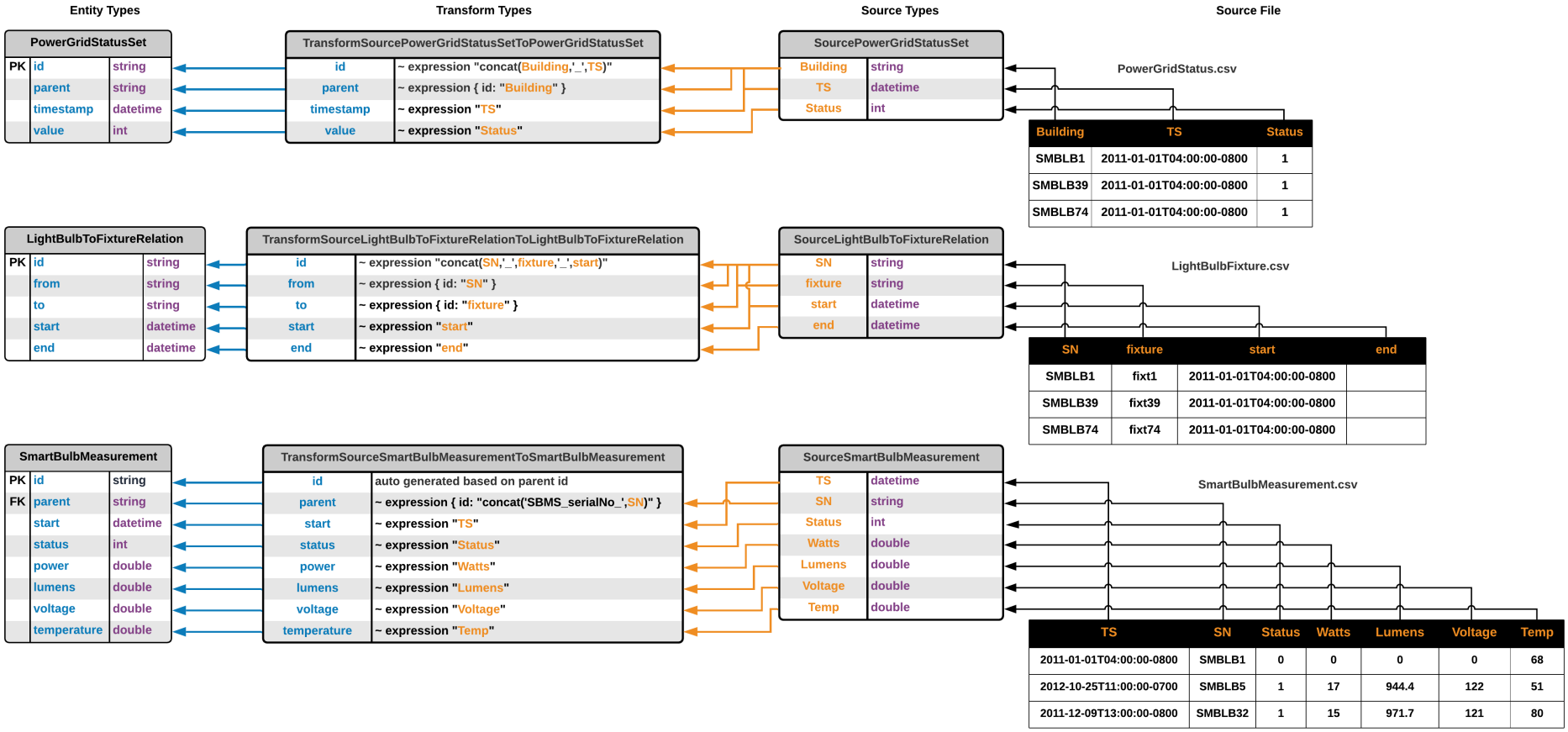
```
{
    "name": "MyCMetric",
    "id": "MyCMetric",
    "expression": "MyExpression"
}
```

LightBulb ERD



LightBulb Data Integration Pipelines




Entity Types
Transform Types
Source Types
Source File
PowerGridStatusSet
PK id string
parent string
timestamp datetime
value int
TransformSourcePowerGridStatusSetToPowerGridStatusSet
id ~ expression "concat(Building,'_',TS)"
parent ~ expression { id: "Building" }
timestamp ~ expression "TS"
value ~ expression "Status"
SourcePowerGridStatusSet
Building string
TS datetime
Status int
PowerGridStatus.csv
Building TS Status
SMBLB1 2011-01-01T04:00:00-0800 1
SMBLB39 2011-01-01T04:00:00-0800 1
SMBLB74 2011-01-01T04:00:00-0800 1
LightBulbToFixtureRelation
PK id string
from string
to string
start datetime
end datetime
TransformSourceLightBulbToFixtureRelationToLightBulbToFixtureRelation
id ~ expression "concat(SN,'_',fixture,'_',start)"
from ~ expression { id: "SN" }
to ~ expression { id: "fixture" }
start ~ expression "start"
end ~ expression "end"
SourceLightBulbToFixtureRelation
SN string
fixture string
start datetime
end datetime
LightBulbFixture.csv
SN fixture start end
SMBLB1 fixt1 2011-01-01T04:00:00-0800
SMBLB39 fixt39 2011-01-01T04:00:00-0800
SMBLB74 fixt74 2011-01-01T04:00:00-0800
SmartBulbMeasurement
PK id string
FK parent string
start datetime
status int
power double
lumens double
voltage double
temperature double
TransformSourceSmartBulbMeasurementToSmartBulbMeasurement
id auto generated based on parent id
parent ~ expression { id: "concat('SBMS_serialNo_',SN)" }
start ~ expression "TS"
status ~ expression "Status"
power ~ expression "Watts"
lumens ~ expression "Lumens"
voltage ~ expression "Voltage"
temperature ~ expression "Temp"
SourceSmartBulbMeasurement
TS datetime
SN string
Status int
Watts double
Lumens double
Voltage double
Temp double
SmartBulbMeasurement.csv
TS SN Status Watts Lumens Voltage Temp
2011-01-01T04:00:00-0800 SMBLB1 0 0 0 0 68
2012-10-25T11:00:00-0700 SMBLB5 1 17 944.4 122 51
2011-12-09T13:00:00-0800 SMBLB32 1 15 971.7 121 80

LightBulb Data Flow