|  |  |  |
| --- | --- | --- |
| Image  Oscar Code Generation | |  | | --- | |  |   Describe and Parse metadata using a simplified language. Apply the metadata to standard templates to produce code.  Kevin J Mackey |

# Introduction

This document is an introduction to the Oscar code generation system (CDS). The Oscar CDS consists of:

* A metadata language
* A metadata parser
* A metadata processor that applies the metadata to templates
* Need-specific custom pyratemp templates to produce code

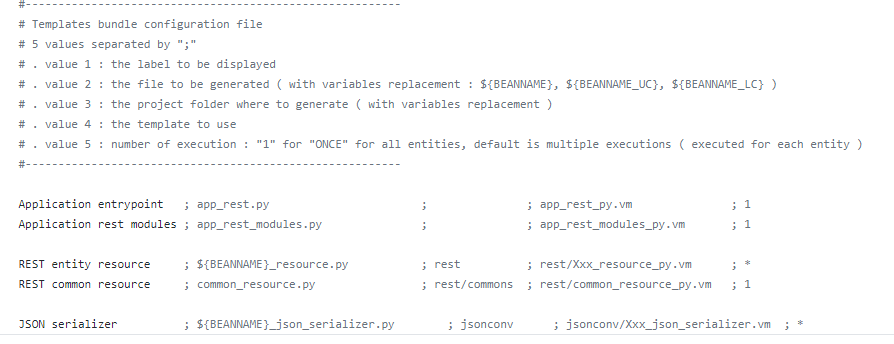
The remainder of this document will cover the above elements.

## Acknowledgments

The Universal Abstract Syntax Tree used in the Oscar CDS is a variation of that described here: <https://pkg.go.dev/github.com/bblfsh/sdk/uast?tab=doc>

The Item, parent-child Item, and Associations are borrowed from concepts laid out in this book: [Vertically Integrated Architectures](https://josjong.com/2018/11/16/my-book-about-vertically-integrated-architectures/)

The configuration of templates being processed once, many times, or files simply being copied is borrowed from the [Telosys](http://www.telosys.org/index.html) code generator, though favoring JSON over semicolon delimited files.



|  |  |
| --- | --- |
|  |  |

# How We Got Here

Data View has been generating code for years. The output of these generation efforts has been code in T-SQL, C#, Java, SSIS.

The templating languages have been XSLT, PYRATEMP, BIML, Jinja, and custom formats.

## Metadata

The metadata to drive the code generation efforts has had various forms and formats. Early versions were results of queries against the INFORMATION\_SCHEMA structures in SQL Server, rendered as XML

Later metadata versions resulted from queries against a Data Dictionary maintained in a SQL Server table, with the queries against the table used to shape the metadata for consumption in templates.

In a further refinement, the metadata was exported to a set of Azure Tables, each structured to meet the needs of specific sets of templates.

BIML can directly query database structures to retrieve the metadata required for BIMLScript templates, and Data View has also used EXCEL, easily maintained by clients, to store very high-level metadata.

## Code Generation Engines

The first of the metadata processors used by Data View were C# programs that applied XSLT templates against XML metadata to produce the resulting code. XSLT, while powerful, proved difficult to scale across teams.

Subsequent engines were Python programs applying either home-grown templating code or PYRATEMP templates against structured metadata. These programs later were migrated to AZURE Serverless Functions (Python) applying structured metadata against the same PYRATEMP templates.

BIML has the advantage of being both the template language and the generation engine, effectively being a generalized engine that uses BIMLScript to both retrieve the metadata and apply it against a template.

## Weaknesses

The two primary weaknesses in these approaches are:

* Custom Metadata
* Custom Engines

### Custom Metadata

For differing code needs, even within the one application, it has been necessary to construct different metadata, shaping it with specific SQL queries, or by storing different metadata in Azure Tables. Each of the metadata structures, SQL table columns, or Azure Table name/value pairs, have been custom in both form and content for each solution.

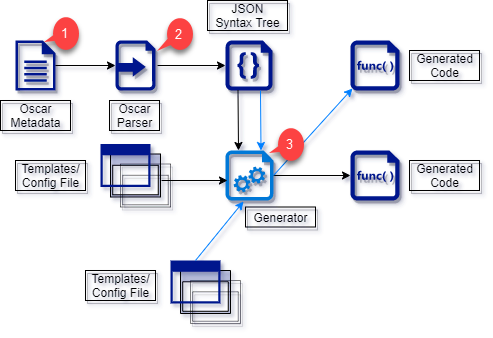
### Custom Engines

The provision of different metadata, and a requirement to deliver differing numbers of output files depending on the application or template needs, has created the need for different metadata-processing programs or functions, even if they can apply the metadata to more than one template. The YRC code generation solution uses nine (9) different Azure Functions and some additional on-prem Python scripts.

The Oscar Code Generation System seeks to simplify the metadata to be used and provides a single generation engine.

# Oscar Code Generation Approach

The diagram below shows the approach taken when generating code using Oscar.

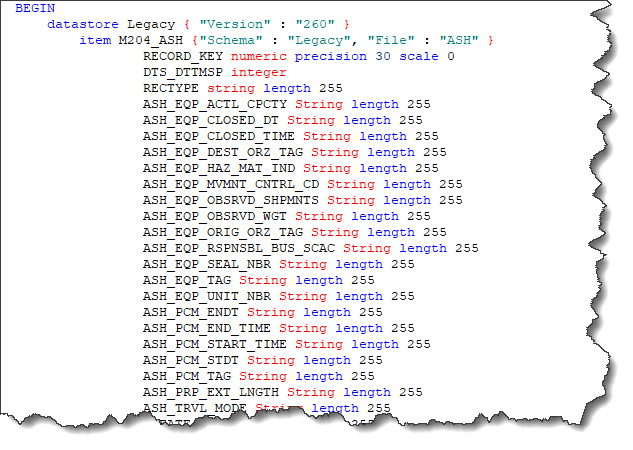


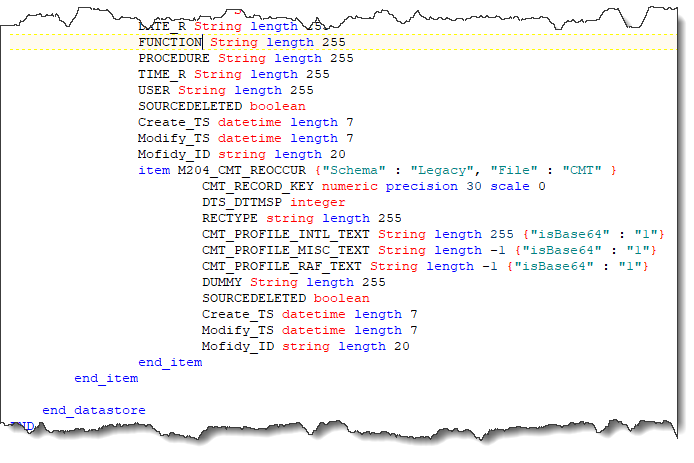
1. Metadata written in the Oscar metadata language
2. Oscar Parser producing an Abstract Syntax Tree (AST) representation of the Oscar metadata in JSON format.
3. Code Generator that reads the JSON AST and merges the data with PYRATEMP templates to produce code.

## Oscar Metadata Language

The Oscar Metadata Language is a standardized, and simplified, notation for describing data stores, such as databases, the items, such as entities in a relational store or nodes in a graph database, stored in such data stores, the relationships between items, and the attributes of these items.

The Oscar metadata language has syntax to concisely describe Datastores, Items, and attributes, as seen in the images below.





The images show a portion of the metadata that could generate code for the SQL Server Service Broker environment at YRC Freight. The sample metadata shows how information such as Data Dictionary Version and table schema might be captured in properties associated with the Datastore to be used, and the Items to be defined.

In addition, the samples show some of the datatypes that the Oscar metadata language can describe.

Finally, the second metadata fragment shows how the parent/child relationship between the M204\_CMT and M204\_CMT\_REOCCUR tables is described. The REOCCUR Item is contained within the M204\_CMT Item definition.

## Oscar Language Elements

The syntax for the Oscar metadata language is straightforward. A block of statements, denotes by a BEGIN/END pair, that contains one or more Datastore/End\_Datastore statements.

The **Datastore/End\_Datastore** statements represent collections of entities such as Relational Database tables, Graph Database nodes, or Collections of Documents in a Document (or Aggregate-Oriented) Database.

Between the Datastore/End\_Datastore statements are one or more **Item/End\_Item** statements that represent the entities/nodes/documents being described.

The Item/End\_Item statements contain the definition of one or more **attributes** corresponding to the properties of the entities/nodes/documents being described.

The Item/End\_Item pairs may also contain child Item/End\_Item pairs which allows modelling parent/child (one to many) relationships.

The Datastore, Item, and Attribute definitions all include these properties:

* Identifier (alphanumeric sequence of characters supplemented by “\_”)
* Display Name (a string literal with text that might be used in system documentation, Web Pages, on-screen forms, database extended properties, a human-readable name for the element being described)
* Properties (a JSON structure of name/value pairs to include any characteristics needing to be captured by the metadata)

## Statements

### Datastore statement

The Datastore statement attributes consist of:

* Datastore keyword
* Identifier
* Display Name (Optional)
* Properties (Optional JSON structure)
* End\_Datastore keyword

These attributes of the Datastore statement become properties of their respective objects during code generation. This is true for Datastore objects, Item objects, and Attribute objects.

The Identifier becomes the object’s Name property and may be referenced by <object>.Name during code generation.

The Display Name attribute becomes the object’s DisplayName property. If no Display Name attribute was supplied in the metadata, the object’s Name property is returned in its stead.

Metadata Properties are collected into a dictionary within the object and are provided to templates via two methods:

* <object>.HasProperty(“Property Name”) which returns True/False
* <object>.GetProperty(“Property Name”) which returns the property’s value, or an empty string

### Item statement

The Item statement consists of these elements:

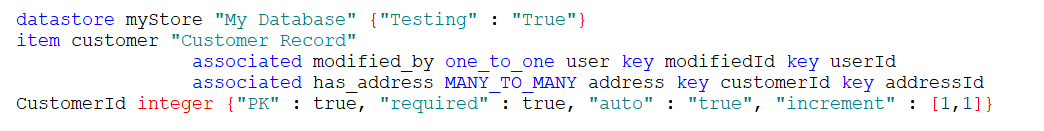
* Item keyword
* Identifier (behaves as for the Datastore statement)
* Display Name (as above)
* Plural (Optional) Can be used if generating code where a plural version of the identifier would be useful (Person/People.) Available at code generation time as <object>.Plural. This returns the Name plus an “s” if the Plural attribute was not included in the metadata.
* Properties (as for the Datastore statement)
* Associated (Optional. This attribute allows for capturing one-to-one and many-to-many relationships between Items)
* Child Item definitions (Optional. These allow for capturing one-to-many relationships between Items)
* End\_Item keyword

### Associated Property

The Associated property is optional and may be specified more than once. It consists of these attributes:

* Associated keyword
* Identifier (a name for this relationship)
* Cardinality (ONE\_TO\_ONE or MANY\_TO\_MANY keyword)
* Item Identifier (name of the associated Item)
* Key Identifier (identifier of the “from” key in this Item)
* Key Identifier (identifier of the “to” key in the associated Item)

Below is an example of both one to one and many to many associations with a Customer Item:



In the one to one association above, the modifiedId attribute of the customer Item contains a value from the set of userId values to be found in the user Item.

The mofidied\_by identifier for the one to one association could be used in a template modelling these Items in a graph database as the name for the relationship between the two Items.

For the many to many association shown above, the customerId and addressId attributes of the customer and address Items could be modelled as columns in an intersection table capturing the many to many relationship between the two Items.

### Item Attributes

The attribute definition is the most complex, and flexible, statement in the Oscar metadata language.

Unlike the Datastore and Item statements, the attribute definition does not begin with an “Attribute” keyword, nor does it end with an “End\_Attribute” keyword.

The common elements it shares with the Datastore and Item statements are:

* Identifier
* Display Name
* Properties

The elements unique to the Attribute statement are:

* Datatype (this element is required for the correct definition of an attribute and must be one of a simplified set of values)
  + Boolean
  + Integer
  + Float
  + GUID
  + Character
  + Numeric
  + String
  + Date
  + Time
  + Datetime
* Length (an optional element consisting of the Length keyword followed by an integer value)
* Precision (an optional element consisting of the Precision keyword followed by an integer value)
* Scale (an optional element consisting of the Scale keyword followed by an integer value)
* Default (an optional element consisting of the Default keyword followed by a constant value or a function call such as GETDATE(). Function calls may be nested.)

The data types known to the Oscar metadata language encompass most of the datatypes in use in applications. The templates used to generate the final code from this metadata can add the nuances required for specific programming languages or tasks.

The grammar definition for the Oscar metadata language is shown in Appendix A. The Parser Rules for the Oscar metadata language are shown in Appendix B.

## #Include Directive

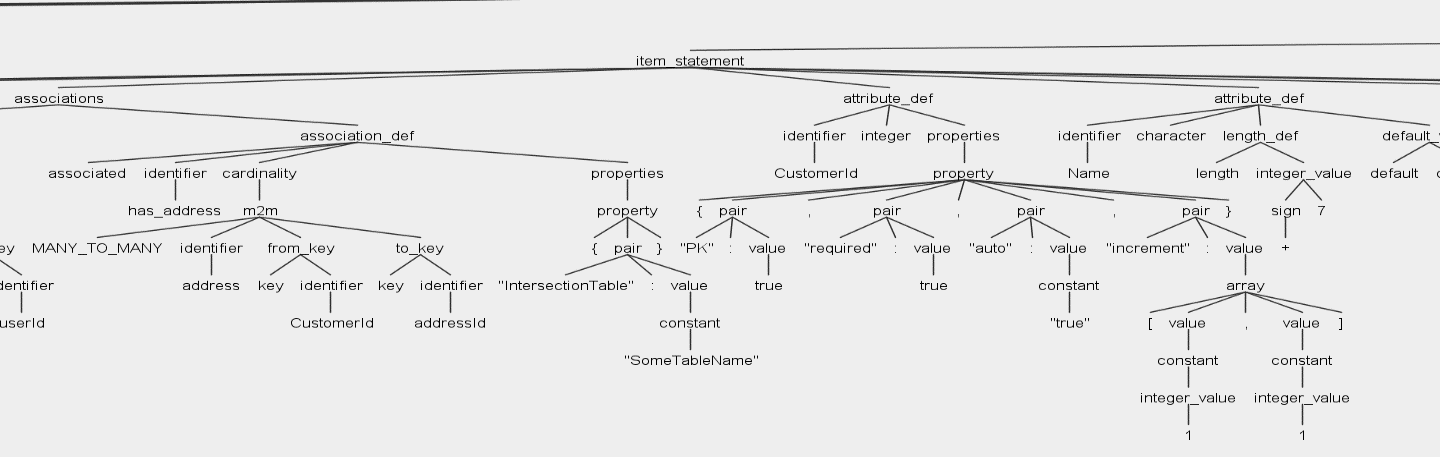
If the first non-whitespace string on a line of Oscar metadata is #include followed by a file path, the Oscar metadata parser will open that file and include the contents at that point in the input stream.

In this way it is possible to split up a metadata model into different files and include only those required to target a specific programming or business problem.

The Oscar metadata files have a .dvo extension.

# Oscar Metadata Parser

The Abstract Syntax Tree (AST) generated by the ANTLR parser (on which the Oscar Metadata Parser is based) is highly detailed and contains structures for each level in the parser rules triggered by the text.



In the image above, commas, colons, brace brackets, and actual language tokens are all included. This syntax tree, while useful for determining if the metadata in the input file is syntactically correct, presents the metadata structures at too low a level to be easily used by code generation templates.

The Oscar Metadata Parser performs a 2nd pass over the AST to group the elements into more easily digested segments. It persists this reduced abstract syntax tree in a JSON file.

This hierarchical structure contains all the information needed to identify the Datastores, Items, Attributes, and interrelationships in the metadata. The representation is standardized allowing for easy, iterative, processing.

The format is a variation on the Universal Abstract Syntax Tree (UAST) that is described here: <https://pkg.go.dev/github.com/bblfsh/sdk/uast?tab=doc>

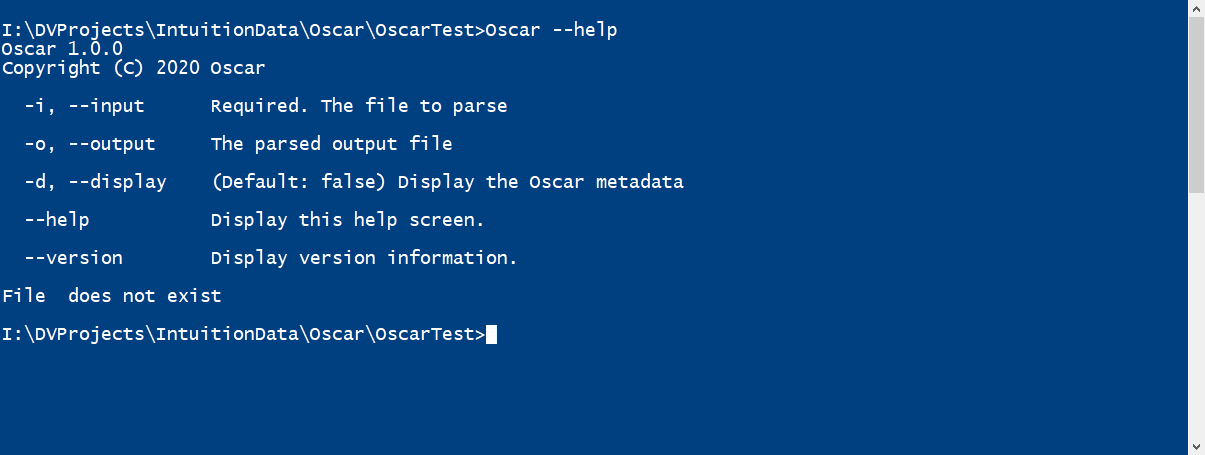
These implementations, both in C# and Python, are simplified and extended versions of the UAST described at the above link.

****

The Oscar Metadata Parser is a C# program that encapsulates a parser built using ANTLR and a Syntax Tree Visitor class that inherits from the template Visitor generated by ANTLR. This Visitor selects the elements of the parser AST to be persisted as JSON. Internally the syntax tree built by the Visitor class is a hierarchical tree of UAST nodes.

UAST nodes in this implementation know how to return their properties as JSON fragments. The persisted JSON representation of the syntax tree combines all the UAST node JSON fragments.

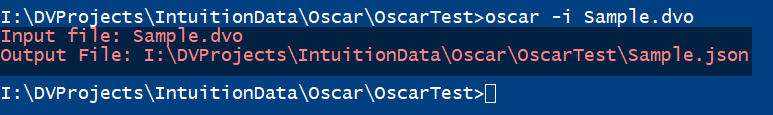
## Oscar Metadata Parser Command Line Options



The image above shows the command line options for the Oscar Metadata Parser.

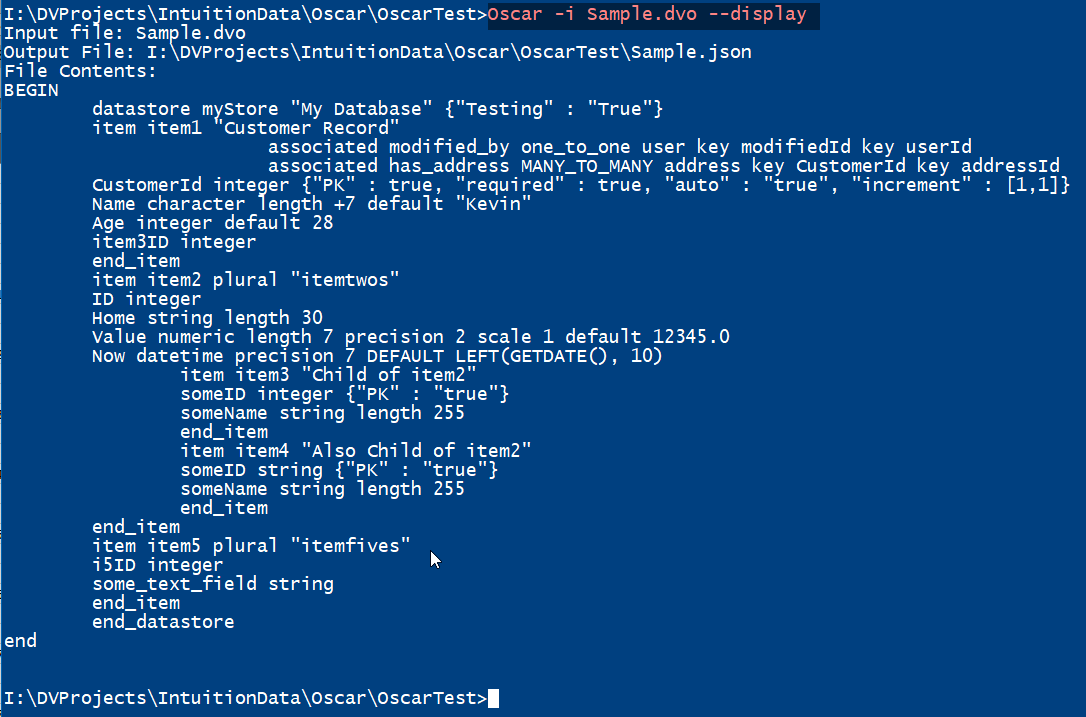
* -i followed by a file path specifies the input metadata file
* -o followed by a file path specifies the output JSON file (if no output file is specified the parser will use the name of the input file with the extension JSON)
* -d if this option is specified the parser will display the contents of the metadata file, including any other files specified with #include

Without the -d or --display modifier, the invocation of the Oscar Metadata Parser looks as below:



When the parser runs it shows the specified input, and the intended output, filenames.

If the -d or --display modifiers are used, the output is:



In all cases, the Oscar Metadata Parser shows the input and output filenames.

Once the metadata is parsed and output to the JSON structure, the work of the Oscar Metadata Parser is complete. The JSON structure is consumed by the Oscar Code Generator and applied to sets of pyratemp templates.

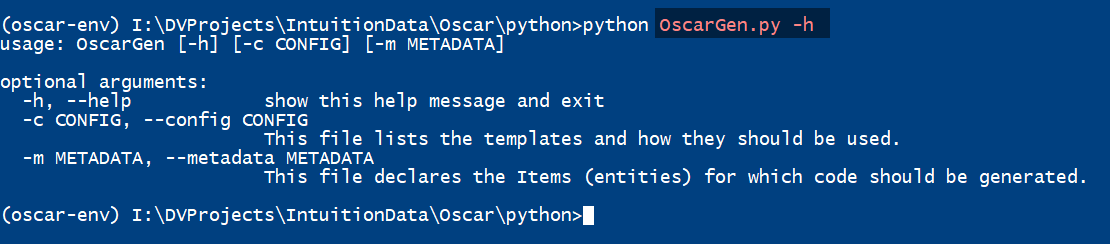
# Oscar Code Generation

The Oscar Code Generator consists of:

* OscarGen.py Python script that loads the JSON metadata and applies these structures against a set of pyratemp templates
* A JSON configuration file (default AppGenConfig.json) that lists the temple files to be used and where the generated output should be stored
* Pyratemp templates addressing the specific generation need

## OscarGen.py

The OscarGen Python script takes up to two (2) command line arguments



* -m –metadata followed by a filepath specifies the JSON metadata file with which to work
* -c –config followed by a filepath specifies the configuration file for this generation run. The default value for this is AppGenConfig.json and the default file is expected to be in the same folder as the OscarGen script)

The OscarGen.py script writes a log of the generation process:

* Loading the JSON metadata
* Start time of the generation process
* Applying each template described in the configuration file
* End time of the generation process

This log is written to OscarGen.log in the folder from which the generation process is invoked.

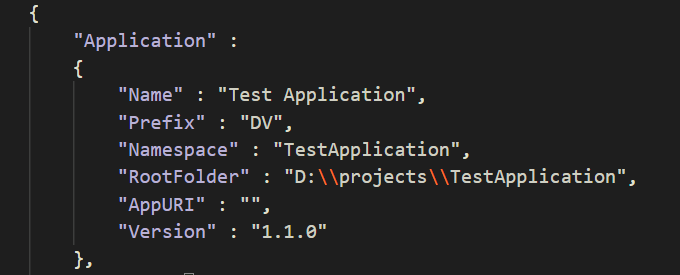
We have seen earlier in this document the structure of the JSON metadata file as produced by the Oscar Metadata Parser.

## Configuration JSON File

The configuration file is a JSON structure broken into two main categories:

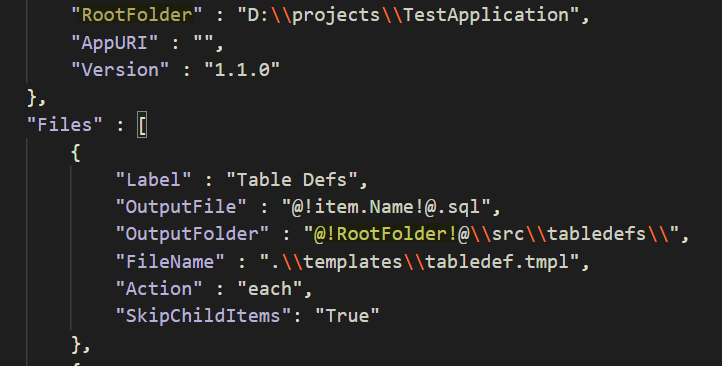
* General information about the application (Name, Root Folder, Version, etc.) under the Application JSON element
* A JSON array under the Files element detailing what pyratemp templates are to be used, where the output should be stored, and whether a template should be processed once for all Items in the metadata, or once per Item.

An example of the Application JSON element is below:



All the elements that are part of the Application JSON structure are provided to each of the pyratemp templates during code generation.

There are no predefined elements that must be part of the Application JSON element, though RootFolder or something designating the root application folder is reasonable to provide. This element can be used in the rest of the AppGenConfig file to allow for flexibility in the definition of file sources and destinations during code generation.



As seen above, the RootFolder element can be used as part of the definition of the output folder for generated files.

The substitution is made using pyratemp placeholders, the same placeholders used in the code generation templates themselves.

## Files Array

Unlike the Application element, which may contain whatever elements a generation scenario requires, the Files array has set of known elements it may contain. Each element of the array may contain:

|  |  |
| --- | --- |
| Element | Description |
| Label | A human-readable name for the template being described. This value is written to the log file when the template is being processed. |
| FileName | The file path of the template file. |
| OutputFile | The filename of the generated code. It is possible, as in the example above, for pyratemp substitutions to be made to create different files during code generation. |
| OutputFolder | The folder path for the generated output files. Pyratemp substitutions may be made to alter where the output file(s) will be written. |
| Include | A JSON array of Item names to be included during generation, should a subset of metadata items be required. This element is optional. |
| Exclude | A JSON array of Item names to be skipped during code generation. This element is optional. |
| SkipChildItems | If this element is present in the configuration file child items will be omitted during code generation. This ensures only Items with their own Child Items are sent to the pyratemp processor.  This may be used where the generated code for a Parent Item contains also the code for the Child Item (XMLShredders, for example.) |
| Action | The permissible values for this element are   * COPY * ONCE * EACH |

The Action element requires further explanation.

### Copy

The Oscar Code Generator will copy files with the COPY action from the FileName filepath to the OutputFolder\OutputFile filepath without alteration.

This element is used for resources such as images, fixed text values, code elements implementing application-specific functionality not amenable to generation.

### Once

Template files marked with the ONCE action are processed exactly once for all the Items in the metadata. In practical terms this means an array of Items is passed to the template and the template should be coded such that it iterates over the **items** array.

Only one output file will be created from this application of the template.

### Each

Here the template file is called for each Item in the metadata file and an individual output file will be created for each application of the template. The content of the Item object being processed is passed to the template in the **item** element.

## Code Generation

The OscarGen.py script loads the JSON abstract syntax tree and builds a set of arrays of objects. These objects are used in during code generation by the pyratemp templates.

These objects are:

* Datastore
* Item
* Attribute
* Association

These properties are common to all these objects

|  |  |
| --- | --- |
| Property | Description |
| Name | The identifier used in the Oscar Metadata file |
| InternalType | A values used within the Oscar code generation system to identify the metadata element type (e.g. dvo:Item, dvo:Attribute, etc.) |
| Token | Synonym for the Name property |
| Roles | An array of the Universal Abstract Syntax Tree (UAST) roles performed by this metadata element in a particular metadata file. |
| HasRole(“Role”) | Returns True if this metadata element performs the specified Role |
| HasProperty(“Property”) | Returns True if this metadata element contains the specified property |
| GetProperty(“Property”) | Returns the value of the specified property or an empty string |

The Name property can be retrieved by the pyratemp templates using the form @!<object>.Name!@. The other properties are accessed in the same way.

## Datastore Element Properties

|  |  |
| --- | --- |
| Property | Description |
| DisplayName | Returns the contents of the Display Name metadata element or the Name property if Display Name was not specified. |
| GetProperty(“Property”) | Returns the value of the specified property or an empty string |
| HasProperty(“Property”) | Returns True if this metadata element contains the specified property |
| HasRole(“Role”) | Returns True if this metadata element performs the specified Role |
| InternalType | A values used within the Oscar code generation system to identify the metadata element type (e.g. dvo:Item, dvo:Attribute, etc.) |
| Name | The identifier used in the Oscar Metadata file |
| Roles | An array of the Universal Abstract Syntax Tree (UAST) roles performed by this metadata element in a particular metadata file. |
| Token | Synonym for the Name property |

## Item Element Properties

|  |  |
| --- | --- |
| Property | Description |
| Associations | Returns an array of Association objects defined for this Item. |
| AttributeCount | Returns the count of Attributes defined for this Item. |
| Attributes | Returns an array of Attribute objects defined for this Item. |
| CapName | The Name property with the initial letter capitalized and the remaining letters converted to lowercase. Often used for, for example, object class names in C# or Java. |
| CapPlural | Returns the Plural property with an initial capital letter and the remainder of the value in lowercase. |
| ChildItems | Returns an array of child Item objects. |
| Datastore | Returns the Datastore object corresponding to the parent Datastore of this Item. |
| DisplayName | Returns the contents of the Display Name metadata element or the Name property if Display Name was not specified. |
| FirstAttribute | Returns the first Attribute defined for this Item or a Null value. |
| GetProperty(“Property”) | Returns the value of the specified property or an empty string |
| HasAssociations | Returns a Boolean True/False depending on whether the metadata defines any Associations for this Item. |
| HasAttributes | Returns a Boolean True/False depending on whether the metadata defined any Attributes for this Item. |
| HasChildItems | Returns a Boolean True/False depending on whether the metadata defines any child Items within this Item. |
| HasParentItem | Returns a Boolean True/False depending on whether or not this is a child Item. |
| HasProperty(“Property”) | Returns True if this metadata element contains the specified property |
| HasRole(“Role”) | Returns True if this metadata element performs the specified Role |
| HasSchema | Returns Boolean True/False depending on whether the metadata included a Schema property for this Item. |
| InternalType | A values used within the Oscar code generation system to identify the metadata element type (e.g. dvo:Item, dvo:Attribute, etc.) |
| Name | The identifier used in the Oscar Metadata file |
| Parent | Returns the parent of this Item or a Null value. |
| PKAttributes | Returns an array of Attribute objects that include a “PK” property. |
| Plural | Returns the value of the Plural metadata element if specified or the Name of the Item with an “s” suffix. |
| Roles | An array of the Universal Abstract Syntax Tree (UAST) roles performed by this metadata element in a metadata file. |
| Schema | Returns the value of the Schema property is one was included in the Item metadata or a Null value. |
| Token | Synonym for the Name property |

## Association Element Properties

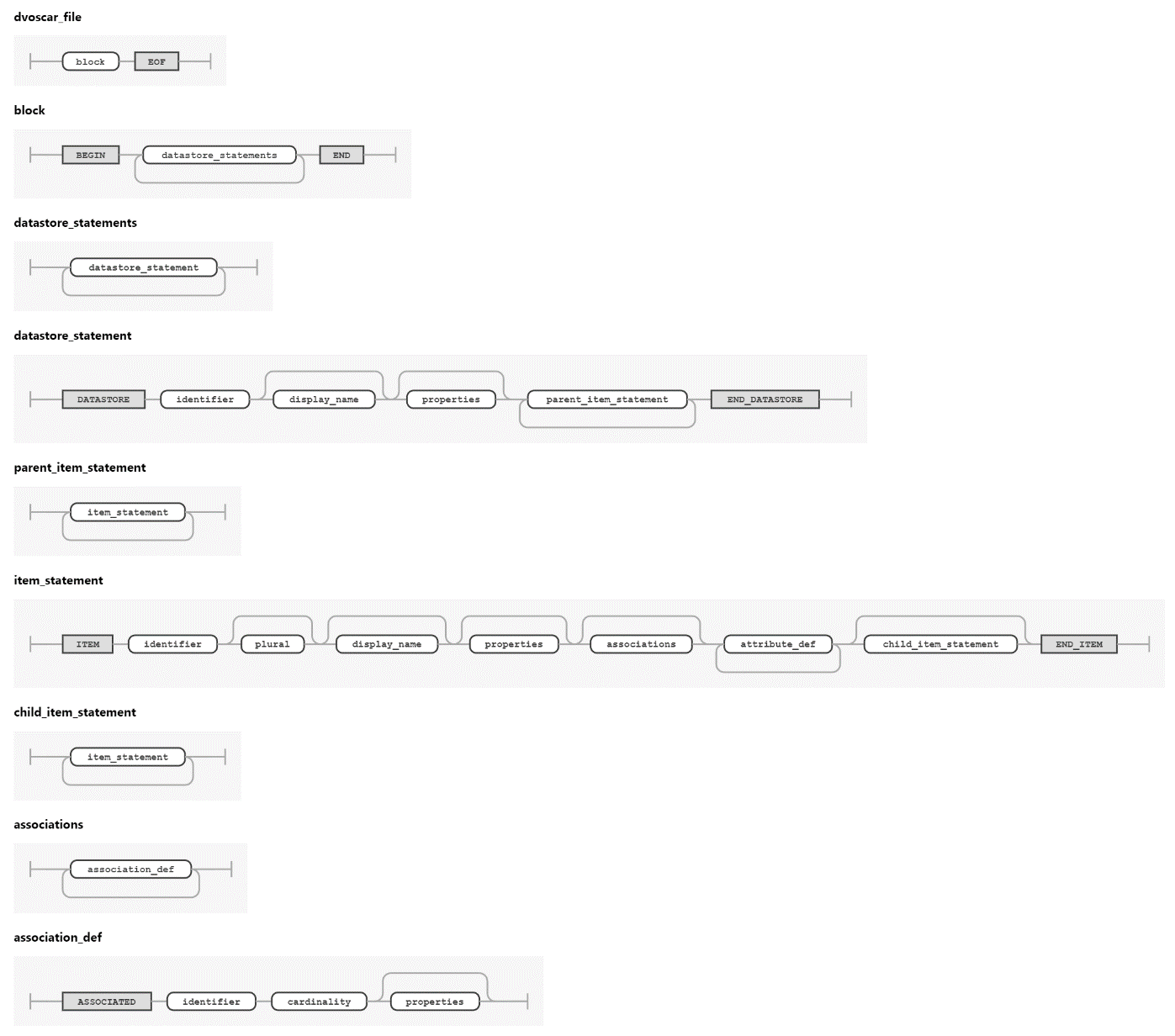
|  |  |
| --- | --- |
| Property | Description |
| Cardinality | Returns "1-2-1" or "M-2-M" depending on the cardinality specified in the metadata for this Association. |
| FromKey | Returns the Name of the Attribute in the parent Item. |
| GetProperty(“Property”) | Returns the value of the specified property or an empty string |
| HasProperty(“Property”) | Returns True if this metadata element contains the specified property |
| HasRole(“Role”) | Returns True if this metadata element performs the specified Role |
| InternalType | A values used within the Oscar code generation system to identify the metadata element type (e.g. dvo:Item, dvo:Attribute, etc.) |
| Item | Returns the Name of the target Item. |
| Name | The identifier used in the Oscar Metadata file |
| Roles | An array of the Universal Abstract Syntax Tree (UAST) roles performed by this metadata element in a particular metadata file. |
| Token | Synonym for the Name property |
| ToKey | Returns the Name of the Attribute in the target Item. |

## Attribute Element Properties

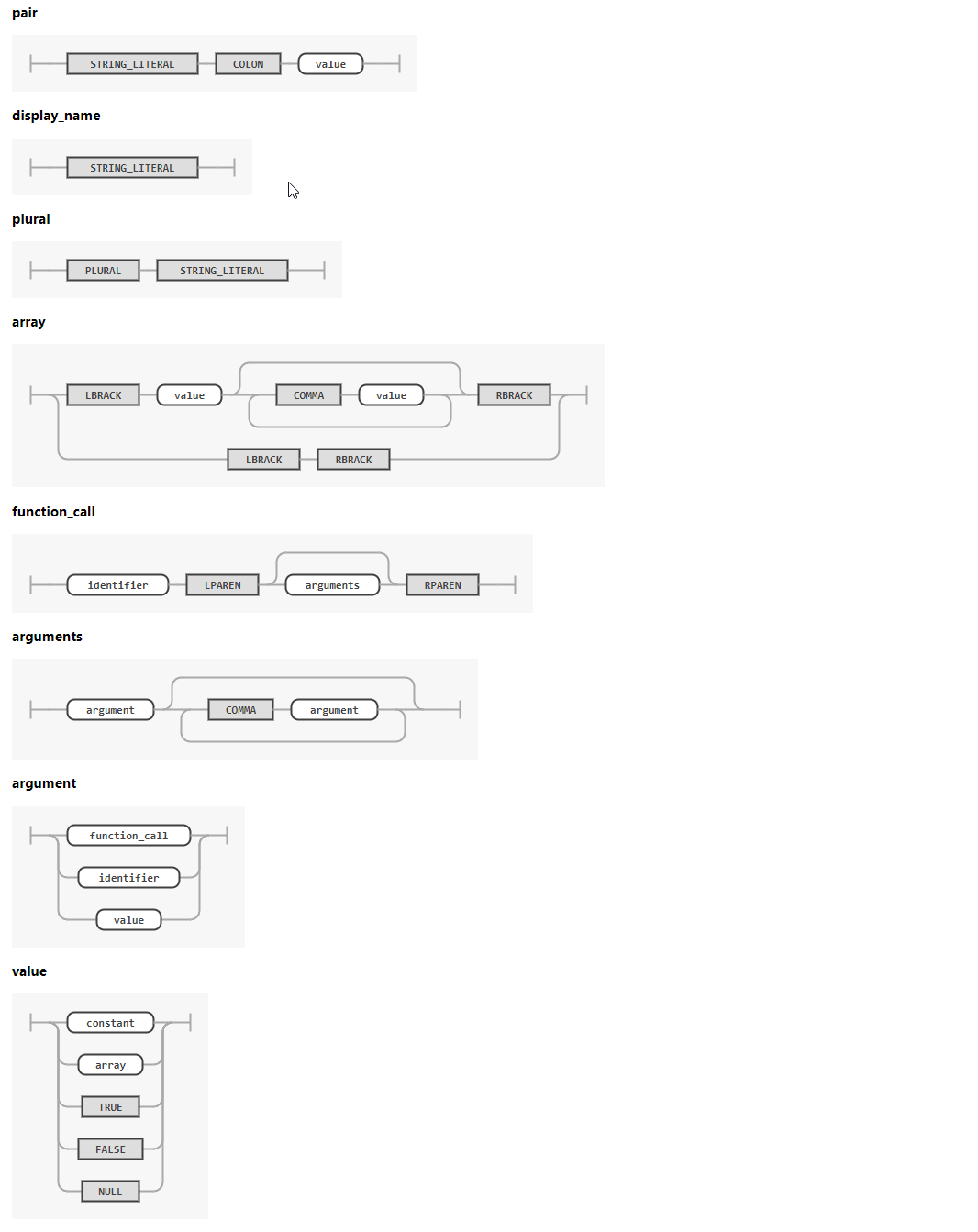
|  |  |
| --- | --- |
| Property | Description |
| Datatype | Returns the datatype of this attribute. |
| Default | Returns the Name of the Attribute in the parent Item. |
| DisplayName | Returns the contents of the Display Name metadata element or the Name property if Display Name was not specified. |
| GetProperty(“Property”) | Returns the value of the specified property or an empty string |
| HasProperty(“Property”) | Returns True if this metadata element contains the specified property |
| HasRole(“Role”) | Returns True if this metadata element performs the specified Role |
| InternalType | A values used within the Oscar code generation system to identify the metadata element type (e.g. dvo:Item, dvo:Attribute, etc.) |
| Name | The identifier used in the Oscar Metadata file |
| Roles | An array of the Universal Abstract Syntax Tree (UAST) roles performed by this metadata element in a particular metadata file. |
| Token | Synonym for the Name property |

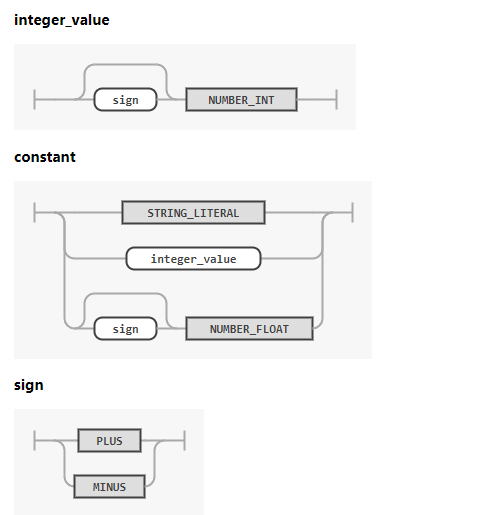
The Length, Precision, Scale, and Default properties are available in the Property array for the Attribute being generated.

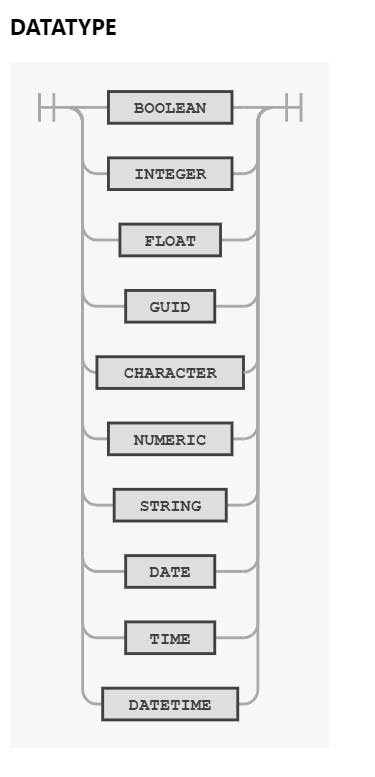
# Appendix A–Oscar Grammar











# Appendix B–Oscar Parser Rules

