A screenshot of a computer

Description automatically generated

https://github.com/esobrino/Datovy.ModelConsole.WinUI

EDAM Interactive Data Model & Visualization Console

# BACKGROUND

In the previous post the effort to create a Graphics Library using SKIA and WINUI 3 as presented to eventually be available as a Web Assembly control that could run in a Browser (see [1]). This post offers an alternative to SKIA using XAML. The motivation to use XAML is based on how easy (less code and effort) is to implement needed functionality to make the diagram interactive and the obvious next step was to try XAML. In XAML a Canvas can have a combination of graphics and UI controls allowing to define graphical object instances that can be tracked, moved, and transformed with less effort than in SKIA.

The rest of the document provides details on how to pass an object instance to be added to the diagram.

# PRIMITIVE OBJECTS

The console will eventually support multiple primitive objects that could be defined using code as shown for the Table primitive that is defined using the TableInfo class:

string schemaName = ENTITY;

ColumnList columns = new ColumnList();

var p = columns.Add(schemaName, "PersonID", 20);

p.IsKey = true;

columns.Add(schemaName, "SexCode", 20);

columns.Add(schemaName, "EthnicityCode", 20);

columns.Add(schemaName, "RaceCode", 20);

columns.Add(schemaName, "BirthDate", type: "DATETIMEOFFSET");

columns.Add(schemaName, "BirthLocationID", 20);

columns.Add(schemaName, "DeathDate", type: "DATETIMEOFFSET");

columns.Add(schemaName, "NationalityCode", 20);

columns.Add(schemaName, "IdentificationNumber", 40);

columns.Add(schemaName, "IdentificationNumberStateCode", 20);

columns.Add(schemaName, "DriverLicenseNumber", 40);

columns.Add(schemaName, "DriverLicenseStateCode", 20);

columns.Add(schemaName, "PassportNumber", 20);

columns.Add(schemaName, "SocialSecurityID", 20);

TableInfo table1 = new TableInfo();

table1.Columns = columns;

table1.SchemaName = ENTITY;

table1.TableName = PERSON;

Alternatively, an instance of TableInfo could be created using a JSON file:

{

"CatalogName": null,

"SchemaName": "Entity",

"TableName": "Person",

"Columns": [

{

"SchemaName": "Entity",

"ColumnName": "PersonID",

"Description": null,

"OrdinalPosition": 0,

"Type": "VARCHAR",

"Size": 20,

"IsNullable": true,

"IsIdentity": false,

"IsKey": true,

"IsForeignKey": false,

"Constraints": []

},

...

{

"SchemaName": "Entity",

"ColumnName": "SocialSecurityID",

"Description": null,

"OrdinalPosition": 0,

"Type": "VARCHAR",

"Size": 20,

"IsNullable": true,

"IsIdentity": false,

"IsKey": false,

"IsForeignKey": false,

"Constraints": []

}

]

}

To draw the tables as shown at the top of the document, fetch the TableInfo instance that contains columns, and constraints as presented below by calling the GetPersonTable(). This example creates an instance of TableInfo and uses DrawTable to define and draw a Table.

// Draw the Entity::Person table

var e1 = Data\_Table\_Entity.GetPersonTable();

var t1 = Table.DrawTable(\_context, 10, 80, 40, e1);

t1.SetBackground(Colors.LightYellow);

model.Add(t1);

// now draw the Entity::PersonName table

var e2 = Data\_Table\_Entity.GetPersonNameTable();

var t2 = Table.DrawTable(\_context, 500, 80, 40, e2);

t2.SetBackground(Colors.Honeydew);

model.Add(t2);

# ORTHOGONAL PATH CONNECTORS

To connect a shape with another it was decided to do using a collection of orthogonal lines (a path) whose vertices will be round corners as shown below:

A close-up of a wire

Description automatically generated

These shapes will be used to specify a relationship between columns of different tables that may represent a relationship or a foreign key. In this version these paths are simple and made up of 3 line segments with 2 end points. The round segments are drawn using a Bezier curve using 3 points. Eventually these segments may be split to add additional lines to allow building complex paths.

How to draw the orthogonal paths in code follows:

GlOrthoPath.Draw(\_context, 10, 600, 100, 800);

GlOrthoPath.Draw(\_context, 200, 600, 110, 800);

GlOrthoPath.Draw(\_context, 210, 600, 300, 800, GlSide.Top);

GlOrthoPath.Draw(\_context, 300, 600, 210, 800, GlSide.Top);

# MOVING THINGS AROUND

Each shape has one or more handling areas that allow the selection its selection, while keeping the pointer (mouse) left-button pressed, the object can be moved around to place it in the required position. Orthogonal shapes handlers are specific points within the path lines that when approached the handling circle appears allowing them to be selected and moved around as explained.

# CONCLUSIONS

By switching to XAML the ability to quickly implement an interactive playground the task to draw graphic and primitive objects become easier to code.

# VERSION 1.0 ROAD MAP

The goal of version 1 is to provide a simple to use graphics playground for developers to quickly load and draw tables and allow them to manage how connectors (orthogonal paths) are placed to show relationships table/columns.

There is a long way to complete the features envisioned for version 1, that include:

|  |  |
| --- | --- |
| **Task** | **Status** |
| Provide to define a Table in code or as a JSON document. | DONE |
| Provide an API endpoint to allow passing a JSON document to be added. | Proposed |
| Define and Draw table primitives. | DONE |
| Define and Draw orthogonal connector shapes. | DONE |
| Select and move shapes around for tables and orthogonal shapes. | DONE |
| Allow an orthogonal path to be reshape interactively by dragging individual line-segments into new positions. | Proposed |
| Allow orthogonal path endpoints to be moved around. | Proposed |
| Allow an orthogonal path line to be split into additional line segments. | Proposed |
| Allow an orthogonal endpoints to have adornments to display the cardinalities of provided constraints. | Proposed |

Although there are not that many features some of those may take some time to implement.

# CONTRIBUTING

Contributions to this effort will help to have a better free and simple to use and extend structure data modeling visualization components. Since (eventually) it will run in the browser as a WebAssembly application it could easily be added to more complex applications.

You can contribute by:

* Offering your critic and/or opinion on the road map, v1 feature set, or whatever you would like to say.
* Help documenting the code or write how-to learning guides.
* Take on a feature.
* Extend/code an existing graphic object, primitive or other to fulfill v1 requirements.
* Move the code to WebAssembly using the uno framework task that should be easy to do and making sure it behaves exactly as its WinUI counterpart.
* Whatever other thing you think you could contribute.

# REFERENCES AND LINKS

[1] see <https://github.com/esobrino/Datovy.ModelConsole.WinUI>