Lessons in Technomancy

Using code generation in a dynamic data system to build flexible and stable games fast.

A. Tito

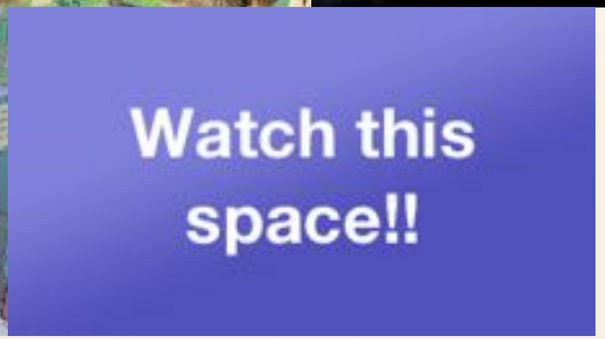
“… If we'd be back in time a thousand years, trying to explain this place to people, they could only accept it in terms of magic.”

“Then perhaps it is magic. The magic of the human heart, focused and made manifest by technology. Every day you here create greater miracles than a burning bush.”

- Captain Sheridan and Elric, Babylon 5:"The Geometry of Shadows"

Anna Tito

Lead Project Technomage & Game Development Valkyrie
Hipster Whale



The way of the “lazy” programmer

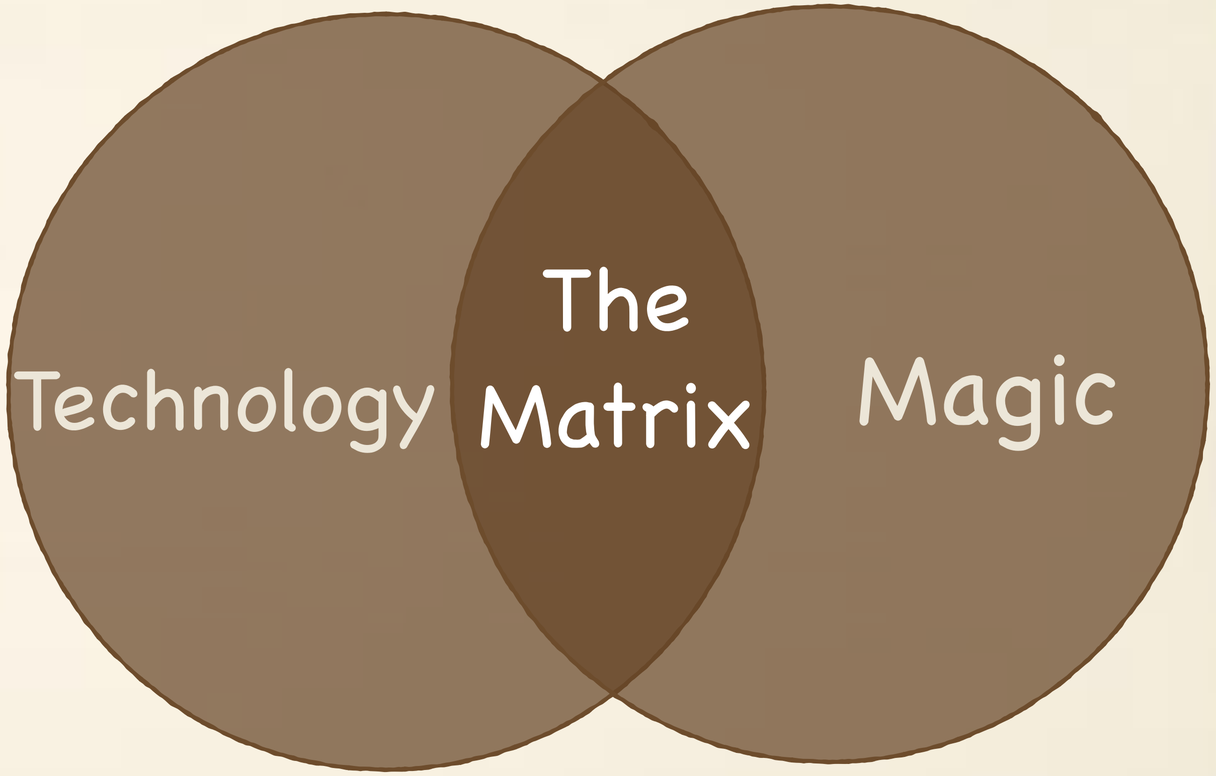
- Game data management
- Tools and pipelines
- Analytics
- Ad integration
- Build Systems
- Asset Management
- In app purchasing
- Social Networking
- On Device Console
- Automated Testing
- Logging
- String manipulation
- Remote Config
- … etc

Write it once, write it right …

Build a toolbox.

What is Technomancy…

- Shared set of assumptions and approaches driven by a core architecture:
  - Flexibility;
  - Reusability;
  - Stability;
  - Write it once;
  - Put the power to create in the hands of those who create….*



* I hate having to wait for builds at 2am because minor tweaks need code changes.

In the beginning was the data…

Data is your backbone

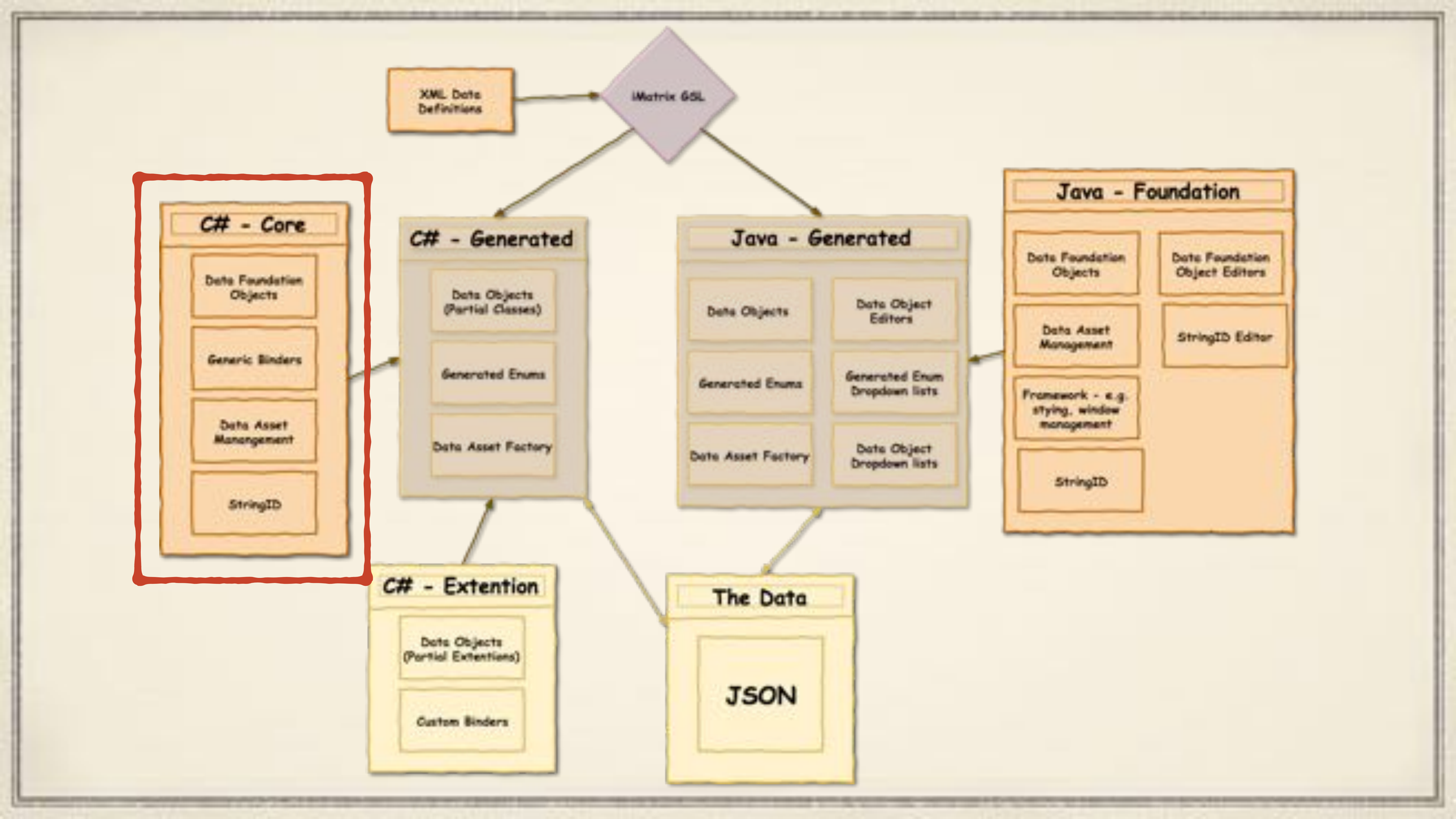
- It needs to drive content and behaviour so you can use it to add new content without new code.
- It needs to be able to be managed by non-programmers.
- It also needs to be:
  - Light weight;
  - Efficient;
  - Easily patched;
  - Flexible; and
  - Robust.

Most data driven games already have schema that drives the data. Why not let it drive the code too?

Bring on the Technomancy…

- The aim of Technomancy
  - Reduce simple errors - maintain stability;
  - Speed up iteration and feature pivots;
  - Support content changes without code changes;
  - Set a strong coding pattern that sets a framework of thinking around how we build the software;
  - Builds a core foundation that allows for data driven flexibility.

Most importantly to appease the code Gods & reduce Anna Rage;


XML Data Definitions
iMatrix GSL
C# - Core
Data Foundation Objects
Generic Binders
Data Asset Management
StringID
C# - Generated
Data Objects (Partial Classes)
Generated Enums
Data Asset Factory
Java - Generated
Data Objects
Data Object Editors
Generated Enums
Generated Enum Dropdown lists
Data Asset Factory
Data Object Dropdown lists
Java - Foundation
Data Foundation Objects
Data Foundation Object Editors
Data Asset Management
StringID Editor
Framework - e.g. styling, window management
StringID
C# - Extention
Data Objects (Partial Extentions)
Custom Binders
The Data
JSON

Core … A Strong Foundation

- Biggest initial time investment in any new language support, yet most important.
- DataManager
  Responsible for managing the loading and unloading data resources
- DataBindingManager & SessionalDataContext
  Responsible for managing the updates of the bound data contexts.
- DataFoundationObject:
  The base object from which all others inherit, this contains any information that is required at all times e.g. the StringID & jsonTypeString.
  - DataObject
    The foundation for all objects with a stand alone file.
  - DataStructure
    The foundation for all objects for all data structures.

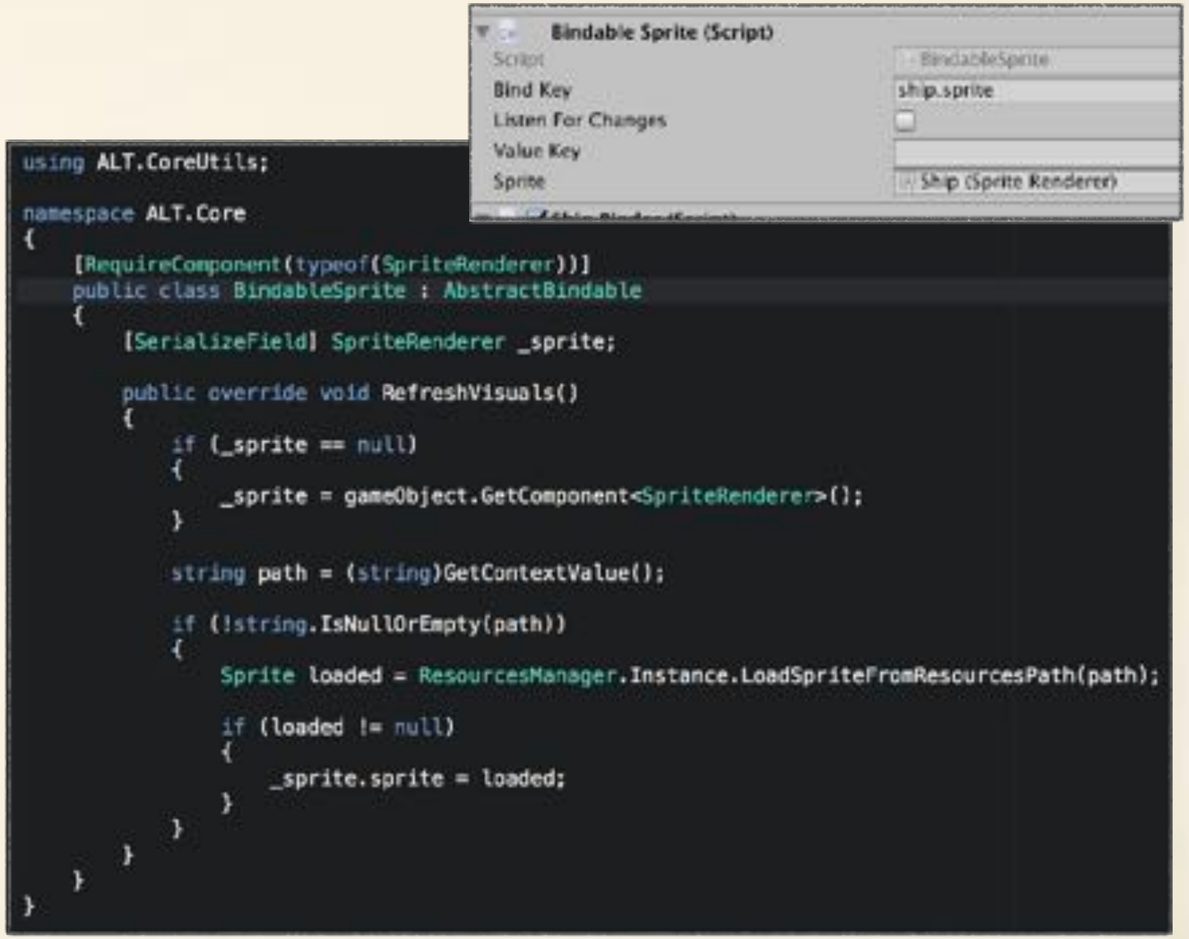
StringID

- Human readable StringID that is internally hashed for quick and efficient equality comparisons.
- For ease of use and reference the human readable ID of a DataObject is the same as its file name;
- Because it is a core concept can also apply some interesting dynamic bindings, e.g the evaluated ID can change based on a variable

```
public StringID(string stringID)
{
    ID_String = stringID;
}

public string ID_String
{
    get
    {
        if (binding && !bindingfound)
        {
            CheckForBindKey(_bindingString);
            _idInt = _idString.GetHashCode();
        }

        return _idString;
    }
    set
    {
        if (String.IsNullOrEmpty(value))
        {
            //error
        }
        else
        {
            if (!CheckForBindKey(value))
            {
                if (value.Contains(" "))
                {
                    string updatedID = value.Replace(" ", string.Empty);
                    Debug.LogError(string.Format("String Ids cannot contain spaces ({0})
                    _idString = updatedID;
                }
                else
                {
                    _idString = value;
                }
            }

            _idInt = _idString.GetHashCode();
        }
    }
}
```

Binders & Binding

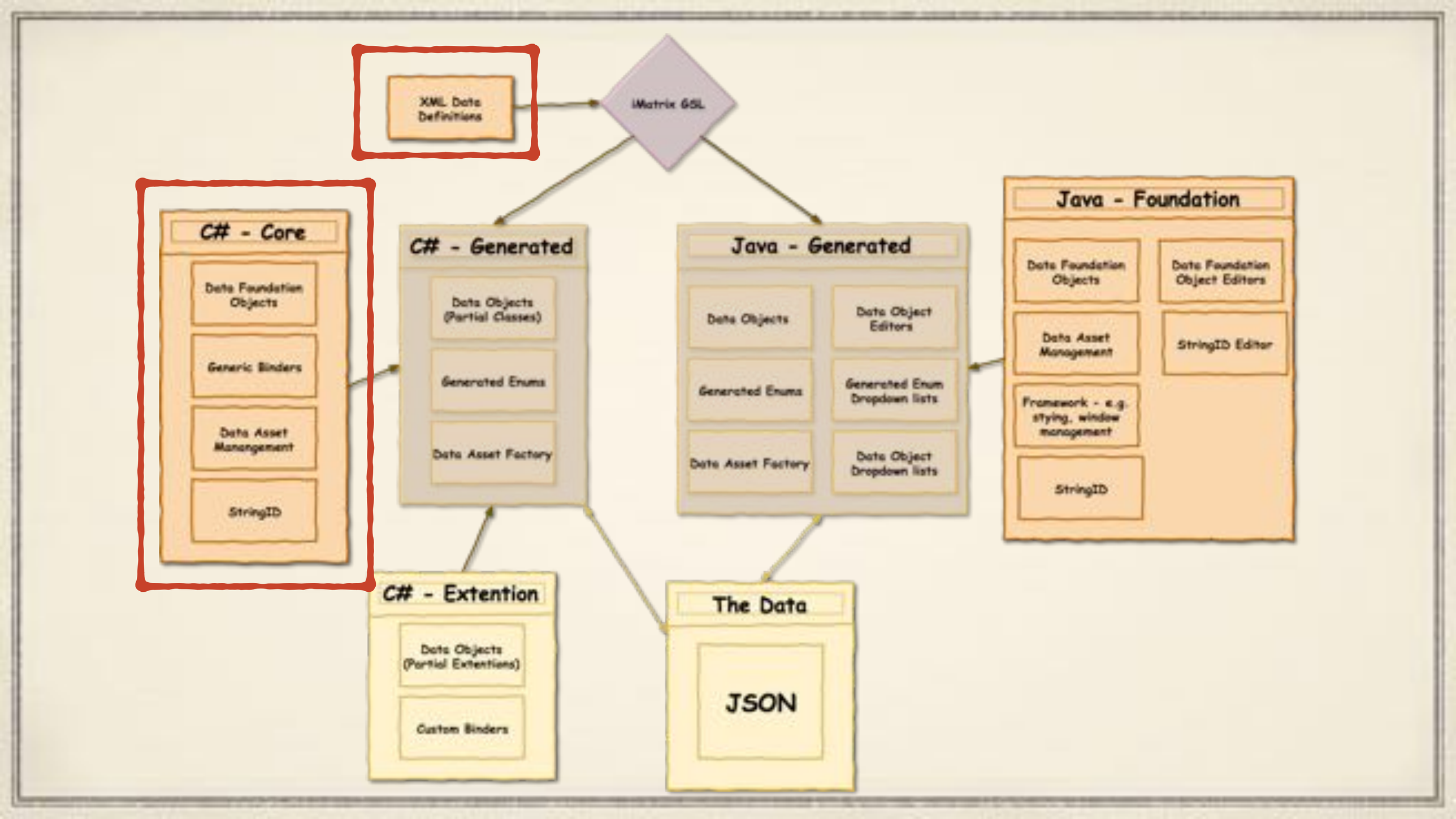
- Bindable Components: inherit from AbstractBindable
  - Image (UI)
  - Sprite
  - ProgressBar
  - Lists of Items
  - String etc.
- Binders: the objects that control the binding, hand written usually pretty light weight, inherit from AbstractBinder<T>



Binders & Binding

- Having the bindability run through the generator means you can:
  - specify any object or field to be bindable.
  - The resource management, update triggers, registration and deregistration all happens in a common base which can be optimised for performance.
  - Bind any bindable field in any data object using the data reference notation e.g
    - ship.sprite
    - fleet.ships.[*].activeHealth
    - Hi my name is {{playerState.firstname}}, and I am going to visit my aunt at the {{Variables.[*].visitAunt.location.name}}

```
public override void BindData(Ship data)
{
    base.BindData(data);

    _bindableSprite.Bind(_rawData);

    _weapons = new DynamicDataObjectPool<WeaponBinder, Weapon>();
    _weapons.Init(_projectilePrefabBase, null, OnProjectileLoaded);

    _activeHealthContextID = SessionalDataContext.BuildContextID(_rawData.ID, ACTIVE_HEALTH_KEY);

    _currentHealth = _rawData.Health;
    SessionalDataContext.Instance.UpdateTempContext(_activeHealthContextID, _currentHealth);
}
```


XML Data Definitions
iMatrix GSL
C# - Core
Data Foundation Objects
Generic Binders
Data Asset Management
StringID
C# - Generated
Data Objects (Partial Classes)
Generated Enums
Data Asset Factory
Java - Generated
Data Objects
Data Object Editors
Generated Enums
Generated Enum Dropdown lists
Data Asset Factory
Data Object Dropdown lists
Java - Foundation
Data Foundation Objects
Data Foundation Object Editors
Data Asset Management
StringID Editor
Framework - e.g. styling, window management
StringID
C# - Extention
Data Objects (Partial Extentions)
Custom Binders
The Data
JSON

XML Schema

- Object
  - A self contained unit, stored in their own file
- Structure
  - Repeated use units e.g. cost
  - Does not require its own file
- Enums
- SchemaVersion

```
<SchemaVersion number="1" />
<!-- CORE-->

<DataStructure name="XYZ_Double">
    <Element name="x" type="double" default="0.0" />
    <Element name="y" type="double" default="0.0" />
    <Element name="z" type="double" default="0.0" />
</DataStructure>
<DataStructure name="RBGA_Int">
    <Element name="r" type="int" default="255" optional="true"/>
    <Element name="g" type="int" default="255" optional="true" />
    <Element name="b" type="int" default="255" optional="true" />
    <Element name="a" type="int" default="255" optional="true" />
</DataStructure>
<DataObject name="StringDictionary">
    <Element name="stringList" type="array" of="KeyedString" />
</DataObject>
<DataStructure name="KeyedString">
    <Element name="value" type="string" />
</DataStructure>
<DataStructure name="Token" hasInheritedTypes="true">
    <Element name="type" type="TokenType" />
    <Element name="value" type="string" />
</DataStructure>
<DataObject name="UserData">
    <Element name="displayName" type="string" />
    <Element name="persistance" type="array" of="PersistentContext" bindable="true"/>
</DataObject>
<DataStructure name="PersistentContext">
   <!-- as ID as - <Element name="contextIdentifier" type="StringID" />-->
    <Element name="persistentDataTokens" type="array" of="Token" bindable="true"/>
</DataStructure>
<!-- rules -->
<DataStructure name="Rule" hasInheritedTypes="true" isAbstract="true">
    <Element name="sourceContext" type="string" />
    <Element name="operator" type="Operators" />
    <Function name="EvaluateRule" return="bool" isAbstract="true" modifier="public"/>
</DataStructure>
<DataStructure name="IntRule" childOf="Rule">
    <Element name="value" type="int" />
```

XML Schema

Elements

- Double
- Int
- Array - Of
- String
- Bool
- StringID
- Generated Enum Types
- Structures
- Objects

Functions

```
<DataObject name="StringDictionary">
    <Element name="stringList" type="array" of="KeyedString" />
</DataObject>
<DataStructure name="KeyedString">
    <Element name="value" type="string" />
</DataStructure>
<DataStructure name="Token" hasInheritedTypes="true">
    <Element name="type" type="TokenType" />
    <Element name="value" type="string" />
</DataStructure>
<DataObject name="UserData">
    <Element name="displayName" type="string" />
    <Element name="persistance" type="array" of="PersistentContext" bindabl
</DataObject>
<DataStructure name="PersistentContext">
   <!— us ID as - <Element name="contextIdentifier" type="StringID" />—->
    <Element name="persistentDataTokens" type="array" of="Token" bindable="
</DataStructure>
<!— rules —>
<DataStructure name="Rule" hasInheritedTypes="true" isAbstract="true">
    <Element name="sourceContext" type="string" />
    <Element name="operator" type="Operators" />
    <Function name="EvaluateRule" return="bool" isAbstract="true" modifier=
```

XML Schema

Object & Structure Attributes

- Name
- HasInheritedTypes
- isAbstract
- ChildOf

Function Attributes

- Name
- Return
- IsAbstract
- Modifier

Element Attributes

- Name
- Type (array also has ‘of’)
- Default
- Bindable
- Optional

Enum Attributes

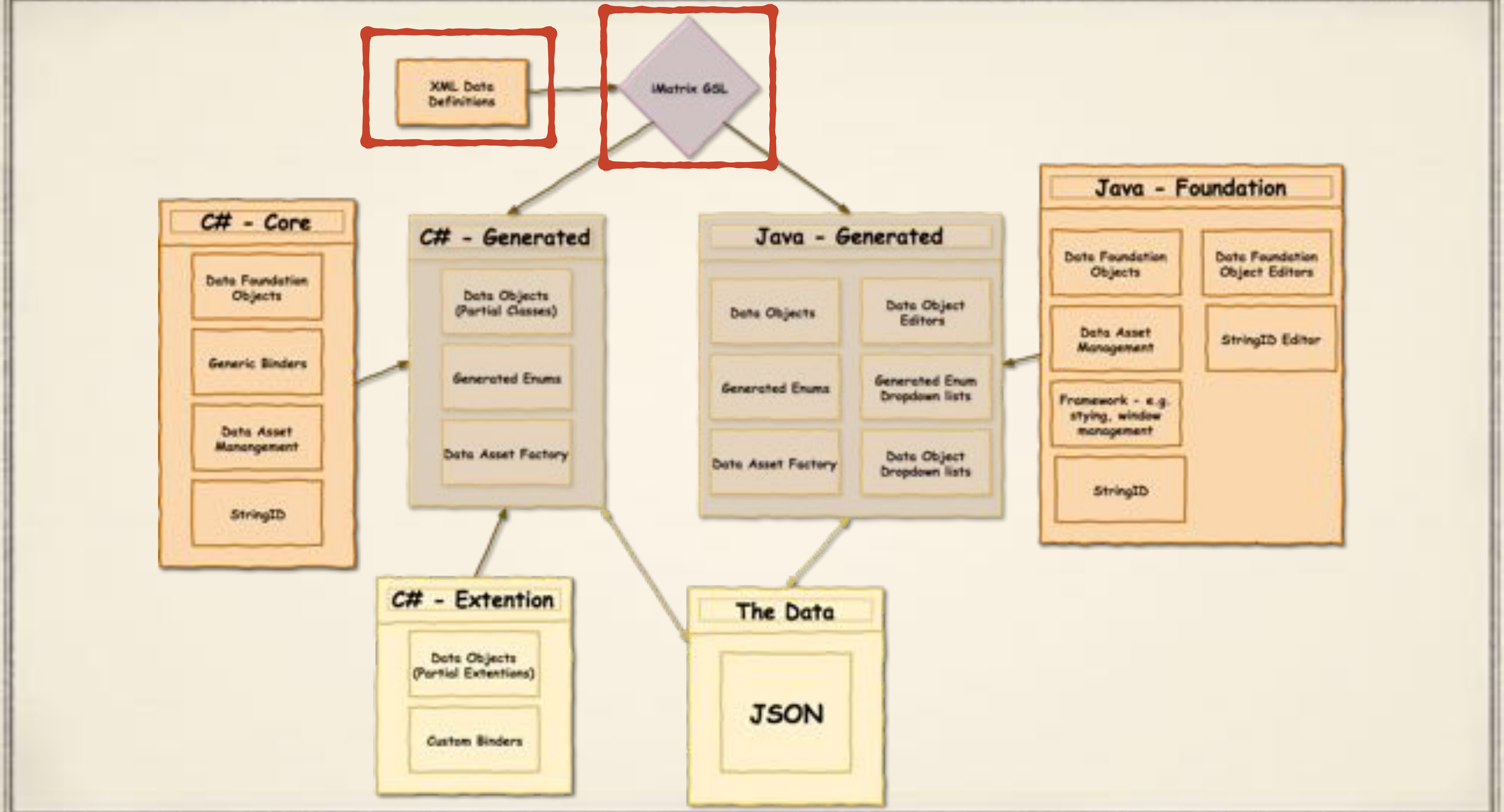
- Name
- Serialize

Enum Items

- Name
- Value (int)

```
<!-- rules -->
<DataStructure name="Rule" hasInheritedTypes="true" isAbstract="true">
    <Element name="sourceContext" type="string" />
    <Element name="operator" type="Operators" />
    <Function name="EvaluateRule" return="bool" isAbstract="true" modifier="public"/>
</DataStructure>
<DataStructure name="IntRule" childOf="Rule">
    <Element name="value" type="int" />
</DataStructure>
```

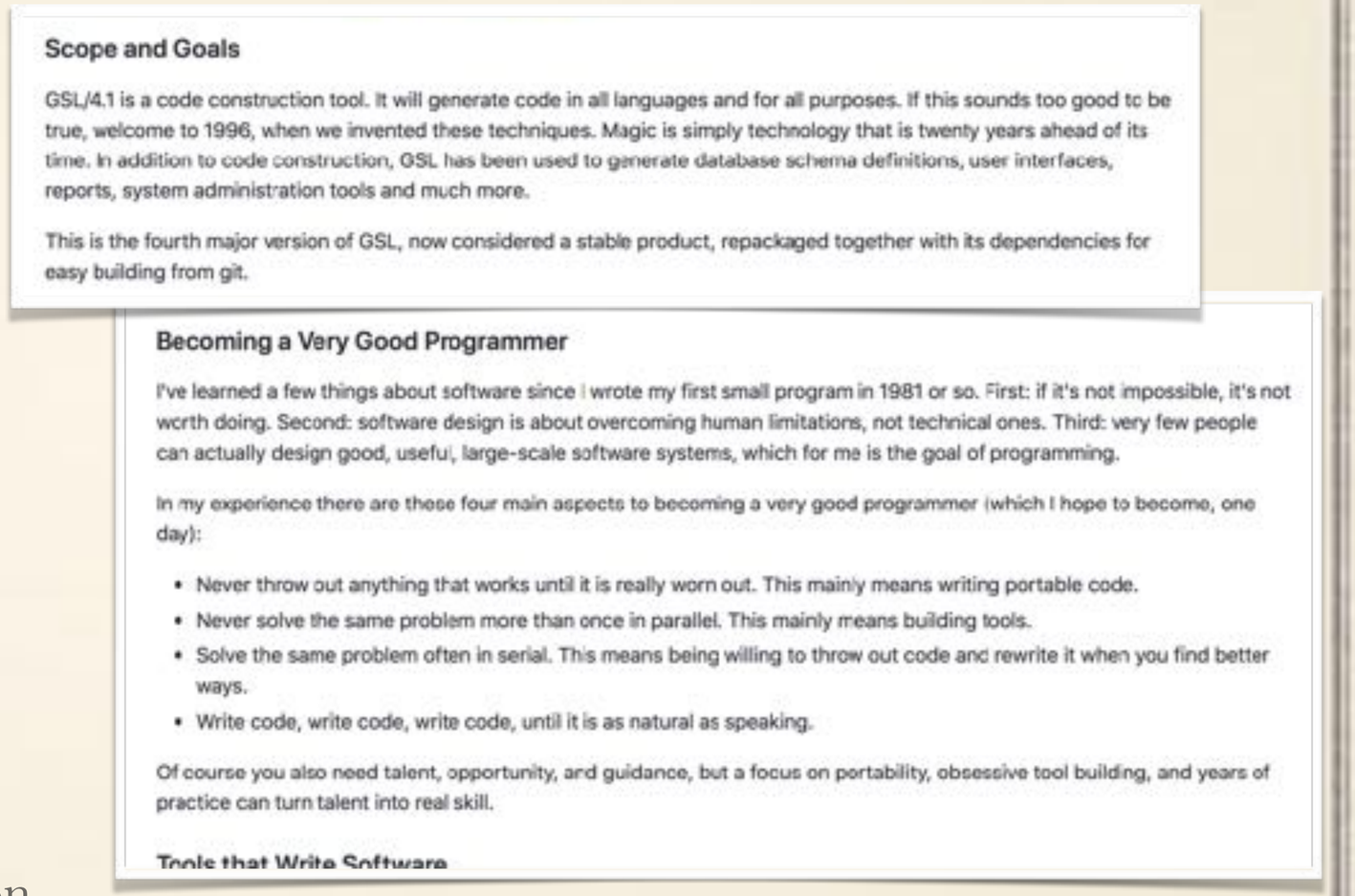
```
<Enum name="TokenType" serialize="true">
    <Item name="String" />
    <Item name="Integer"  />
    <Item name="Float" />
    <Item name="StringID" />
    <Item name="Boolean" />
</Enum>
```


XML Data Definitions
iMatrix GSL
C# - Core
Data Foundation Objects
Generic Binders
Data Asset Management
StringID
C# - Generated
Data Objects (Partial Classes)
Generated Enums
Data Asset Factory
Java - Generated
Data Objects
Data Object Editors
Generated Enums
Generated Enum Dropdown lists
Data Asset Factory
Data Object Dropdown lists
Java - Foundation
Data Foundation Objects
Data Foundation Object Editors
Data Asset Management
StringID Editor
Framework - e.g. styling, window management
StringID
C# - Extention
Data Objects (Partial Extentions)
Custom Binders
The Data
JSON

The Generator: GSL 4.1

GSL/4.1 - a Universal Code Generator:
https://github.com/imatix/gsl

- I know it;
- Parses XML;
- Supports iteration through elements;
- Supports attribute access;
- Has functions, iteration, logic flow;
- Outputs to file;
- Custom runtime XML; and
- It has wonderfully snide documentation.



Scope and Goals

GSL/4.1 is a code construction tool. It will generate code in all languages and for all purposes. If this sounds too good to be true, welcome to 1996, when we invented these techniques. Magic is simply technology that is twenty years ahead of its time. In addition to code construction, GSL has been used to generate database schema definitions, user interfaces, reports, system administration tools and much more.

This is the fourth major version of GSL, now considered a stable product, repackaged together with its dependencies for easy building from git.

Becoming a Very Good Programmer

I've learned a few things about software since I wrote my first small program in 1981 or so. First: if it's not impossible, it's not worth doing. Second: software design is about overcoming human limitations, not technical ones. Third: very few people can actually design good, useful, large-scale software systems, which for me is the goal of programming.

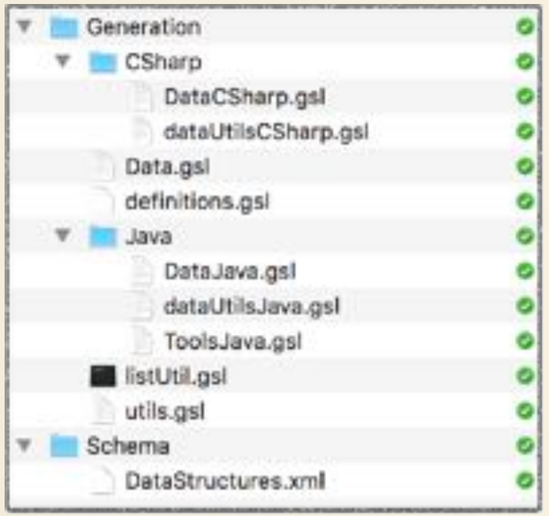
In my experience there are these four main aspects to becoming a very good programmer (which I hope to become, one day):

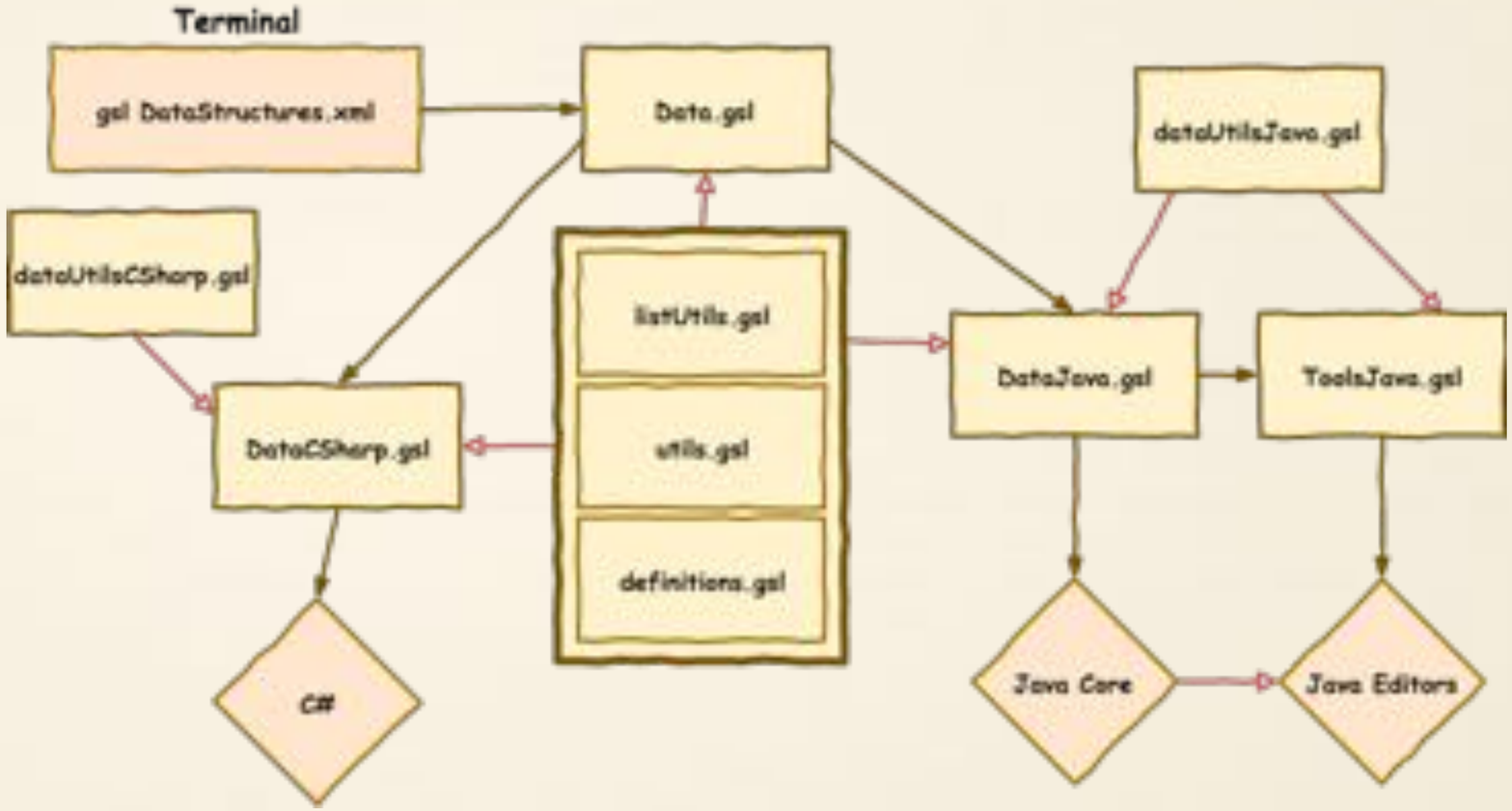
- Never throw out anything that works until it is really worn out. This mainly means writing portable code.
- Never solve the same problem more than once in parallel. This mainly means building tools.
- Solve the same problem often in serial. This means being willing to throw out code and rewrite it when you find better ways.
- Write code, write code, write code, until it is as natural as speaking.

Of course you also need talent, opportunity, and guidance, but a focus on portability, obsessive tool building, and years of practice can turn talent into real skill.

Tools that Write Software

Generation


Generation
CSharp
DataCSharp.gsl
dataUtilsCSharp.gsl
Data.gsl
definitions.gsl
Java
DataJava.gsl
dataUtilsJava.gsl
ToolsJava.gsl
listUtil.gsl
utils.gsl
Schema
DataStructures.xml


Terminal
gsl DataStructures.xml
Data.gsl
dataUtilsJava.gsl
dataUtilsCSharp.gsl
listUtils.gsl
utils.gsl
definitions.gsl
DataCSharp.gsl
DataJava.gsl
ToolsJava.gsl
C#
Java Core
Java Editors

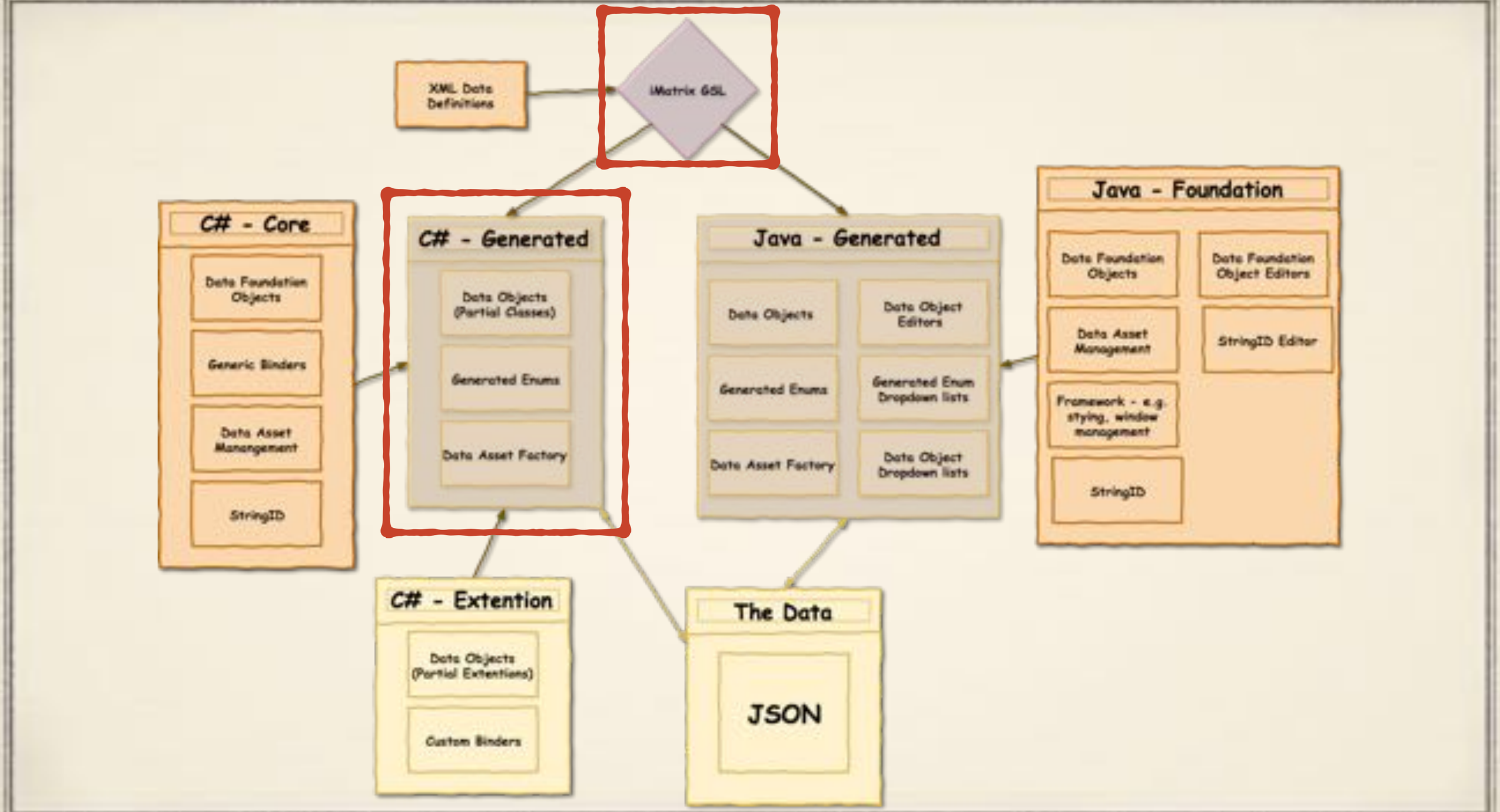
```
#region Json Serializers
        public override string JsonSerialize()
        {
            Dictionary<string,System.Object> dict = new Dictionary<string,System.Object>();
            this.JsonInheritedSerialize(ref dict);

            return  Json.Serialize(dict);
        }

        protected override void JsonInheritedSerialize(ref Dictionary<string, System.Object> dict)
        {

            base.JsonInheritedSerialize(ref dict);
            dict.Add(DataFoundationObject.KEY_SCHEMAVERSION, DataAssetFactory.SCHEMA_VERSION);

            if(!string.IsNullOrEmpty(_sprite))
            {
                dict.Add("sprite", _sprite);
            }
            dict.Add("health", _health);
            if(_type != GeneratedEnums.ShipType.BASIC && _type != GeneratedEnums.ShipType.UNDEFINED)
            {
                dict.Add("type", _type.ToString());
            }
            dict.Add("maxSlots", _maxSlots);
            if(_weaponSlots != null && _weaponSlots.Count > 0)
            {
                List<string> weaponSlotsList = new List<string>();
                for(int i = 0; i < _weaponSlots.Count; i++)
                {
                    weaponSlotsList.Add(_weaponSlots[i].JsonSerialize());
                }
                dict.Add("weaponSlots", weaponSlotsList.ToArray());
            }

        }

        public override void JsonDeserialize(Dictionary<string,System.Object> dict)
        {
            if(dict == null)
            {
                return;
            }

            base.JsonDeserialize(dict);

            if (dict.ContainsKey("sprite"))
            {
                _sprite = dict["sprite"] as string;
            }

            if (dict.ContainsKey("health"))
```

```
function OutputInheritedSerializeFunction  (dataObject, isDataObject)
>       protected override void JsonInheritedSerialize(ref Dictionary<string, System.Object> dict)
>       {
if defined(my.dataObject.useage) & my.dataObject.useage = "runTimeOnly"
>           Debug.LogError("Runtime Data Objects do not serialize to Json");
>           return;
else
>
>           base.JsonInheritedSerialize(ref dict);
>           dict.Add(DataFoundationObject.KEY_SCHEMAVERSION, $(global.DataAssetFactory:Pascal).SCHEMA_V
>
#echo "******* OutputInheritedSerializeFunction for $(my.dataObject.name:) *******"
    for my.dataObject.Element as element
        #echo "******* element: $(element.name:) - $(element.type:)"
        isDataObject = IsTypeDataObject(element.type, "OutputInheritedSerializeFunction sing") | IsKeyT

        if isDataObject
            if defined(element.default)
    >           if(_$(element.name:Camel) != null&& !_$(element.name:Camel).Equals($(element.default:))
            else
    >           if(_$(element.name:Camel) != null)
            endif
    >           {
    >               dict.Add("$(element.name:Camel)", _$(element.name:Camel).ID_String);
    >           }
    >
        elsif  element.type = "array"
            isVectorDataObject = IsTypeDataObject(element.of, "OutputInheritedSerializeFunction vect")
            if IsBaseType(element.of)
    >           if(_$(element.name:Camel) != null && _$(element.name:Camel).Count > 0)
    >           {
    >               dict.Add("$(element.name:Camel)", _$(element.name:Camel).ToArray());
    >           }
    >
            elsif IsTypeEnum(element.of)
    >           if(_$(element.name:Camel) != null && _$(element.name:Camel).Count > 0)
    >           {
    >               List<string> $(element.name:Camel)List = new List<string>();
    >               for(int i = 0; i < _$(element.name:Camel).Count; i++)
    >               {
    >                   $(element.name:Camel)List.Add(_$(element.name:Camel)[i].ToString());
    >               }
    >               dict.Add("$(element.name:Camel)", $(element.name:Camel)List.ToArray());
    >           }
            elsif isVectorDataObject
    >           if(_$(element.name:Camel) != null && _$(element.name:Camel).Count > 0)
    >           {
    >               List<string> $(element.name:Camel)List = new List<string>();
```

Hold up! This seems like a lot of work and time investment…… For what?

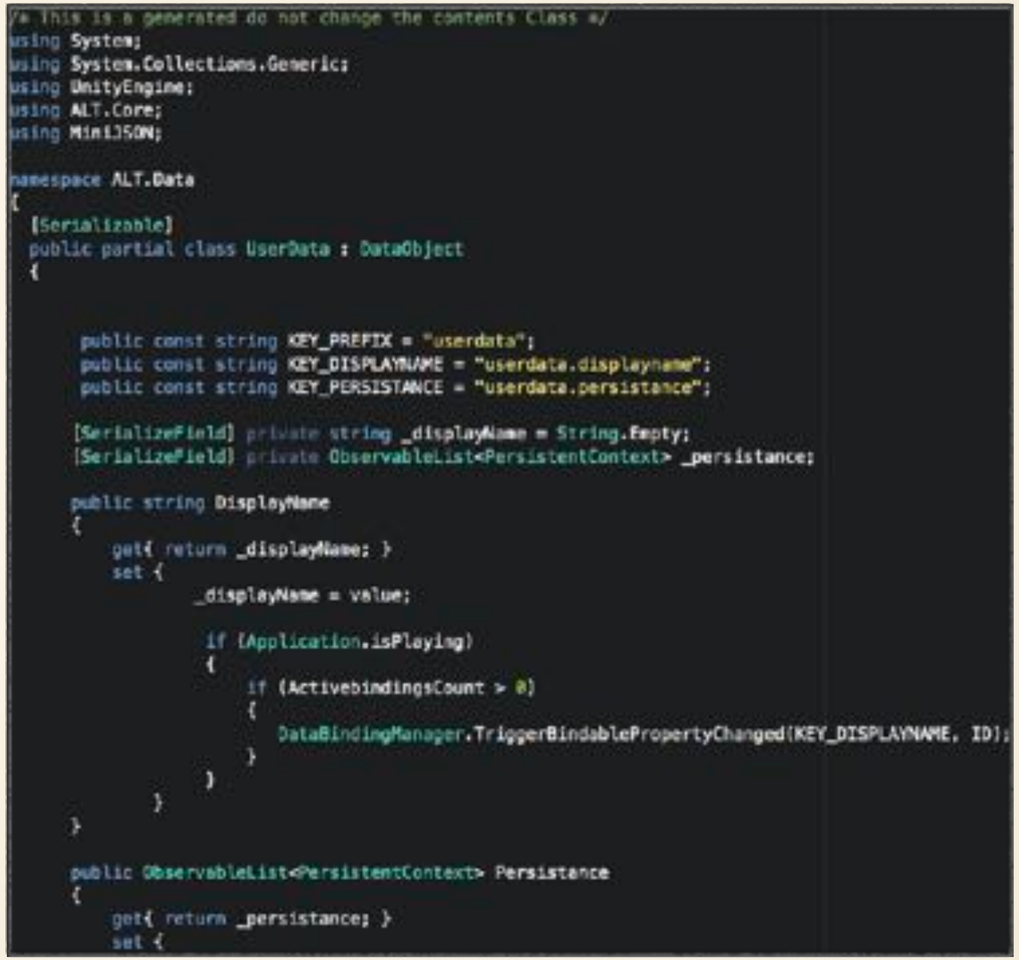
Story Time!

Speedy to pivot & multi-game modes
Adding reusable functionality
Dynamic content
Control, maintainability & stability

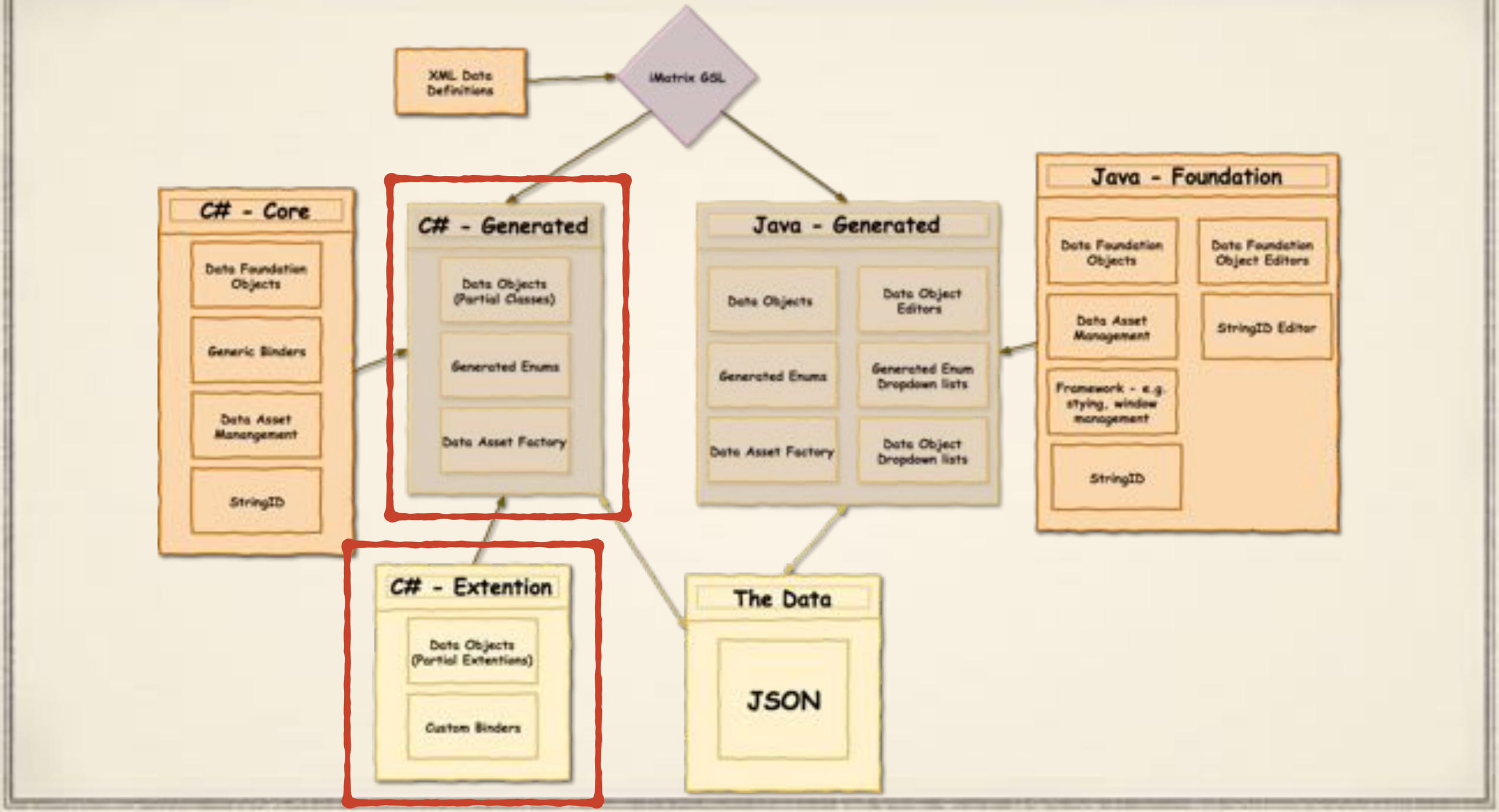

XML Data Definitions
iMatrix GSL
C# - Core
Data Foundation Objects
Generic Binders
Data Asset Management
StringID
C# - Generated
Data Objects (Partial Classes)
Generated Enums
Data Asset Factory
Java - Generated
Data Objects
Data Object Editors
Generated Enums
Generated Enum Dropdown lists
Data Asset Factory
Data Object Dropdown lists
Java - Foundation
Data Foundation Objects
Data Foundation Object Editors
Data Asset Management
StringID Editor
Framework - e.g. styling, window management
StringID
C# - Extention
Data Objects (Partial Extentions)
Custom Binders
The Data
JSON

Generated Core

- Factory
  - A DataObject factory that takes in the type and returns the object
  - Also have jsonType to System.Type support
- Enums
- Classes
  - Inheritance;
  - Variables;
  - Properties;
  - Serialisation & deserialisation;
  - List Manipulation;
  - GetObject functionality for object StringIDs
  - Get Property & Has Property binding including sub data foundation object access; and
  - Safe cloning.



```
/* This is a generated do not change the contents Class */
using System;
using System.Collections.Generic;
using UnityEngine;
using ALT.Core;
using MiniJSON;

namespace ALT.Data
{
  [Serializable]
  public partial class UserData : DataObject
  {


      public const string KEY_PREFIX = "userdata";
      public const string KEY_DISPLAYNAME = "userdata.displayname";
      public const string KEY_PERSISTANCE = "userdata.persistance";

     [SerializeField] private string _displayName = String.Empty;
     [SerializeField] private ObservableList<PersistentContext> _persistance;

     public string DisplayName
     {
         get{ return _displayName; }
         set {
                 _displayName = value;

                 if (Application.isPlaying)
                 {
                     if (ActivebindingsCount > 0)
                     {
                        DataBindingManager.TriggerBindablePropertyChanged(KEY_DISPLAYNAME, ID);
                     }
                 }
             }
     }

     public ObservableList<PersistentContext> Persistance
     {
         get{ return _persistance; }
         set {
```


XML Data Definitions
iMatrix GSL
C# - Core
Data Foundation Objects
Generic Binders
Data Asset Management
StringID
C# - Generated
Data Objects (Partial Classes)
Generated Enums
Data Asset Factory
Java - Generated
Data Objects
Data Object Editors
Generated Enums
Generated Enum Dropdown lists
Data Asset Factory
Data Object Dropdown lists
Java - Foundation
Data Foundation Objects
Data Foundation Object Editors
Data Asset Management
StringID Editor
Framework - e.g. styling, window management
StringID
C# - Extention
Data Objects (Partial Extentions)
Custom Binders
The Data
JSON

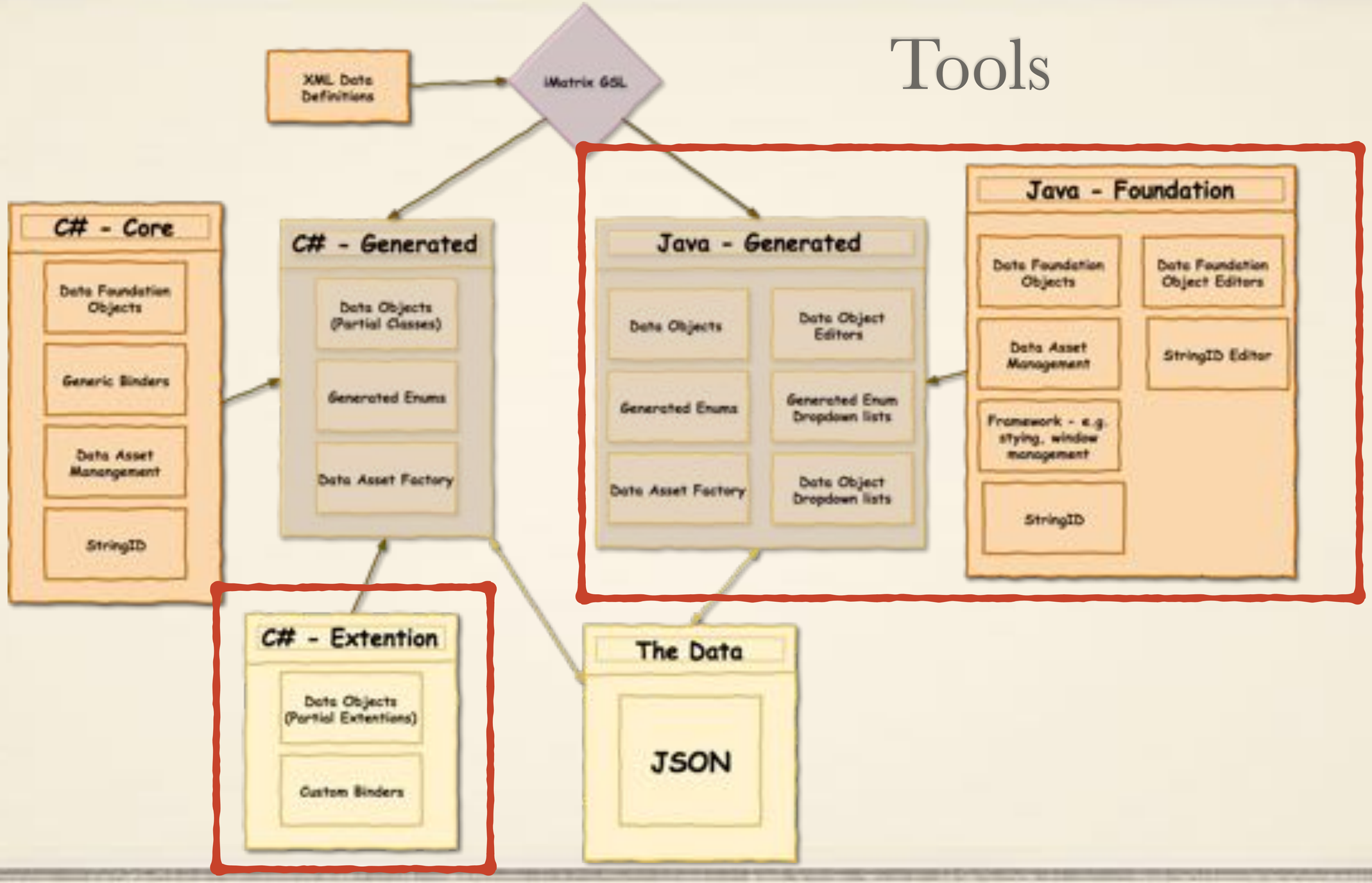
Partial Classes

- C# partial classes are a powerful tool they:
  - Allow the user to extend functionality without touching the generated code; and
  - Allow the programmer to force the existence of certain functions with custom implementations e.g. DataAction children require the PerfromAction() function.

```
/* This is a generated do not change the contents class */
using System;
using System.Collections.Generic;
using UnityEngine;
using ALT.Core;
using MiniJSON;

namespace ALT.Data
{
  [Serializable]
  public partial class PersistentContext : DataStructure
  {
      public const string KEY_PREFIX = "persistentcontext";
      public const string KEY_PERSISTENTDATATOKENS = "persistentcontext.persistentdatatokens";

      [SerializeField] private ObservableList<Token> _persistentDataTokens;

      public ObservableList<Token> PersistentDataTokens
      {
          get{ return _persistentDataTokens; }
          set {
                  _persistentDataTokens = value;
```

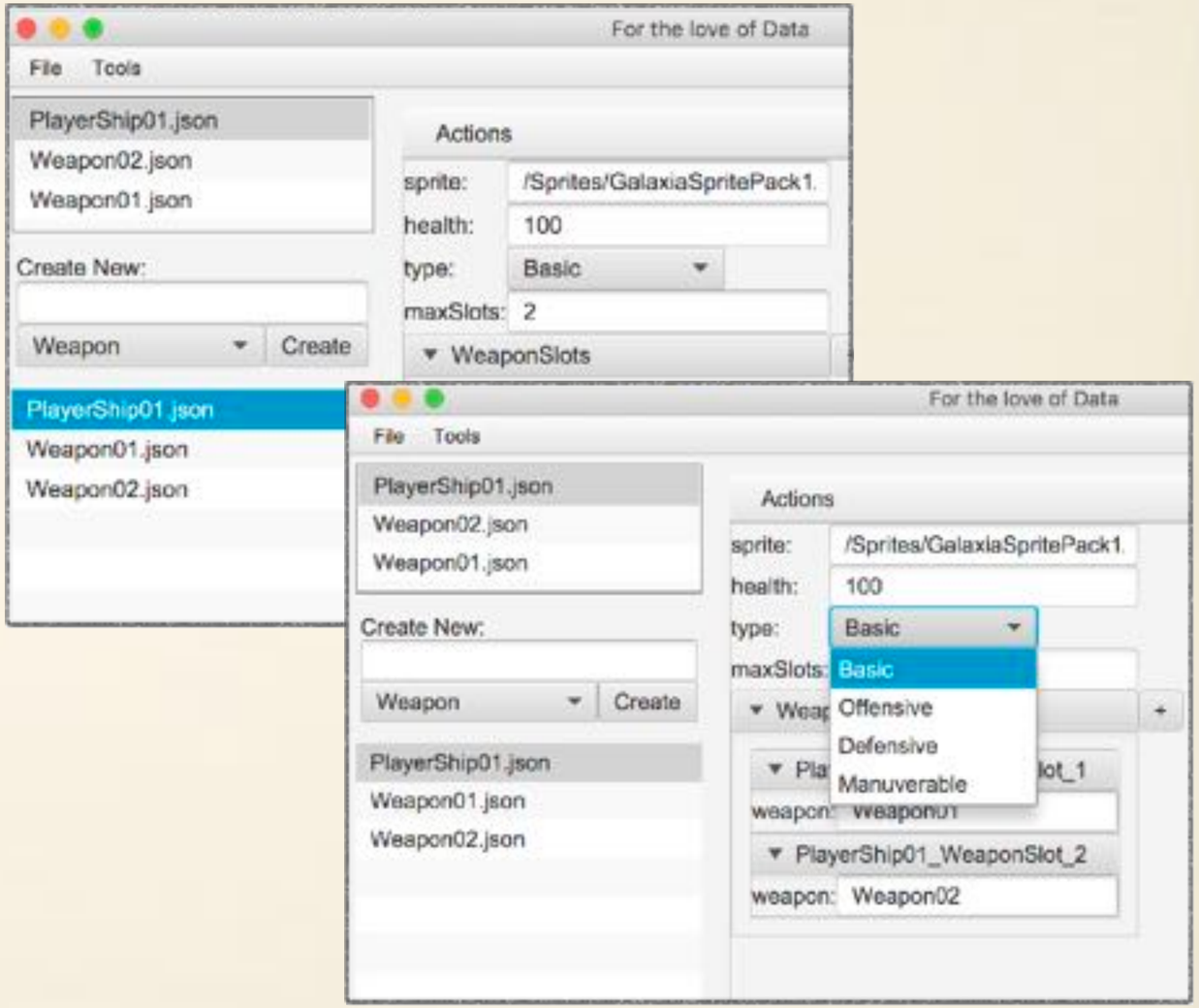
```
using ALT.Core;
using System;

namespace ALT.Data
 {
   public partial class PersistentContext : DataStructure
   {

        public Token GetToken(StringID tokenId, bool addIfDoesntExist = false)
        {
            if (_persistentDataTokens == null || _persistentDataTokens.Count > 0)
            {
                if(addIfDoesntExist)
                {
                    Token token = new Token(tokenId.ID_String);
                    AddPersistentDataTokens(token);
                    return token;
                }
                else
                {
                    return null;
                }
```

Tools


XML Data Definitions
iMatrix GSL
C# - Core
Data Foundation Objects
Generic Binders
Data Asset Management
StringID
C# - Generated
Data Objects (Partial Classes)
Generated Enums
Data Asset Factory
C# - Extention
Data Objects (Partial Extentions)
Custom Binders
Java - Generated
Data Objects
Data Object Editors
Generated Enums
Generated Enum Dropdown lists
Data Asset Factory
Data Object Dropdown lists
Java - Foundation
Data Foundation Objects
Data Foundation Object Editors
Data Asset Management
StringID Editor
Framework - e.g. styling, window management
StringID
The Data
JSON

Tools

- The data editing tools access and work with the same data and schema.
- Generating the tools allow you to:
  - Validate and limit input, reducing data errors.
  - Support custom tooling e.g enum drop downs, type filtering, custom editors for types etc.
  - Generate them out into any language or framework you want. e.g.
    - Java
    - Web tools e.g. React
    - Unity Editor
    - Text editor with intelliSense



Demo*

**please wait while I sacrifice the appropriate poultry to the code and live demo gods*

What else?

- Data validation
- Server-Client message structure
- Server data integration
- Unit tests & default data mocks
- Localisation
- Asset Management (what to bundle & load when)

And so many more things that are on my to do list…

“We are dreamers, shapers, singers, and makers. We study the mysteries of laser and circuit, crystal and scanner, holographic demons and invocations of equations. These are the tools we employ and we know many things.”

-- Elric to Captain Sheridan in Babylon 5:"The Geometry of Shadows"

Questions?

Links

System

- GSL/4.1 - a Universal Code Generator: https://github.com/imatix/gsl
- GSL List library: https://github.com/imatix/gsl/issues/66

Demo Project

- Editor Time Tag Constants Generation, Maxim Kamalov (dogfuntom): https://gist.github.com/dogfuntom/00faf9f3598d845b1a5c
- Unity FSM, Peter Cardwell-Gardner (thefuntastic): https://github.com/thefuntastic/Unity3d-Finite-State-Machine
- Galaxia 2D Space Shooter Sprite Pack #1, Josh Marshall https://www.assetstore.unity3d.com/en/?stay%23!/content/112153#!/content/64944

Anna Tito

@MythicalC