BLACK KNIGHT
FINANCIAL SERVICES
SERVICING & DEFAULT TECHNOLOGIES

Framework Building Blocks

Author: Roberto.Torres@bkfs.com

Date: 11/19/2014

Introduction

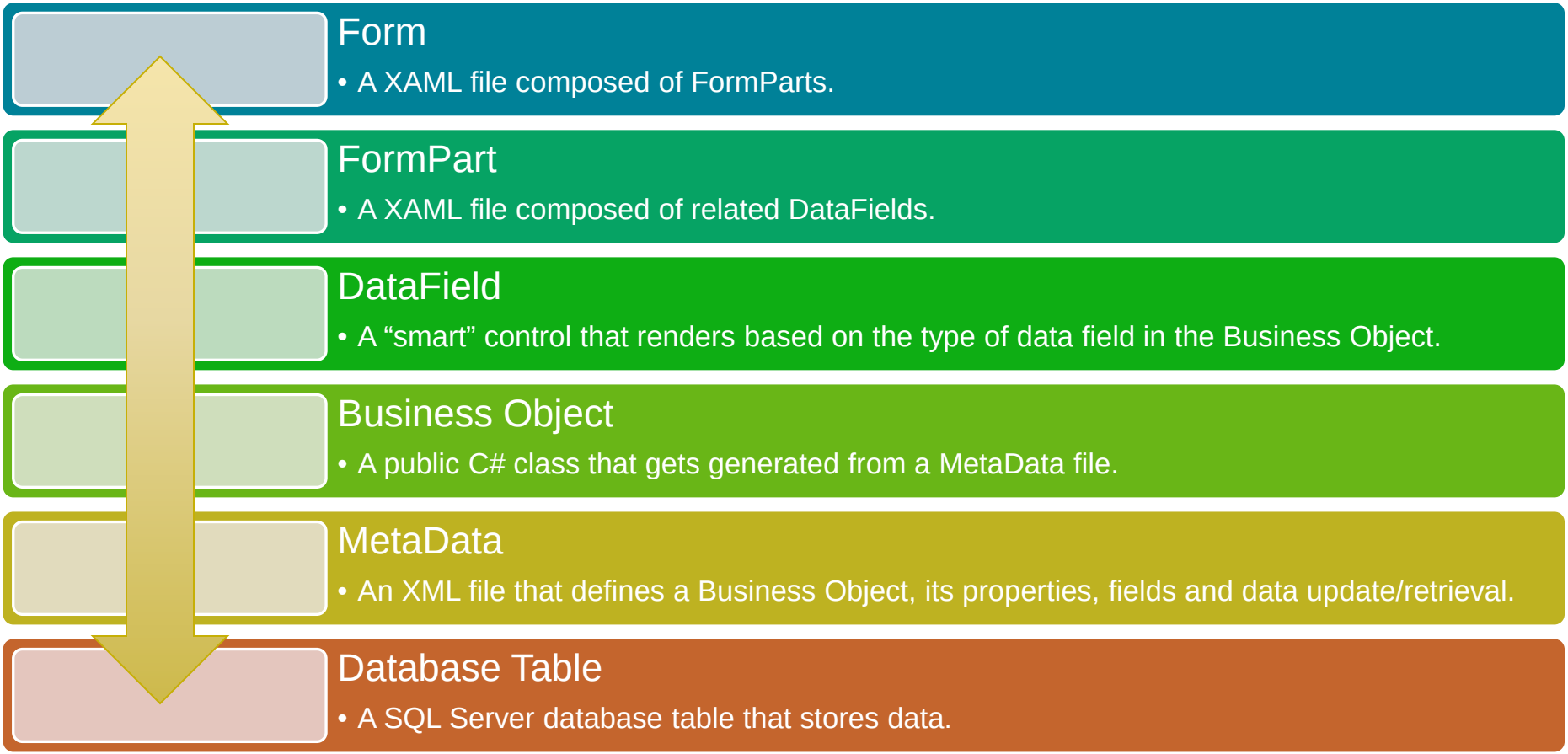
- The purpose of this presentation is to give a simple overview of the LoanSphere (formerly known as “Fusion”) framework.
- Intended audience:
  - Developers that are new to the team.
  - Overly-ambitious Technical Analysts.

BLACK KNIGHT
FINANCIAL SERVICES
SERVICING & DEFAULT TECHNOLOGIES

Chapter 1: Quick Look

From UI to Database

From UI to Database



BLACK KNIGHT
FINANCIAL SERVICES
SERVICING & DEFAULT TECHNOLOGIES

Chapter 2: Deeper Look

From Database to UI

2.1 Database Tables

Database Table

- A SQL Server database table that stores data.

Database Tables - User Schemas

- The Rincon database makes use of User Schemas as a way of organizing complexity.
- Tables are always referred to using their relative schemas: **`[schema name].[table name]`**.
  - This means a name alone can help you understand the usage of a given table.
  - At the time of this writing there are 41 schemas in the database – each with its own set of defined tables.
  - For more on User Schemas, see http://technet.microsoft.com/en-us/library/ms190387(v=sql.105).aspx

Database Tables - Relationships

- Relationships between tables sometimes cross schema lines.
  - **`Mortgage.CollateralProperty`** is a junction table that just holds information specific to the relationship between **`Mortgage.Loan`** and **`Collateral.Property`**.
  - See Figure 2.1.1.

Database Tables - Relationships

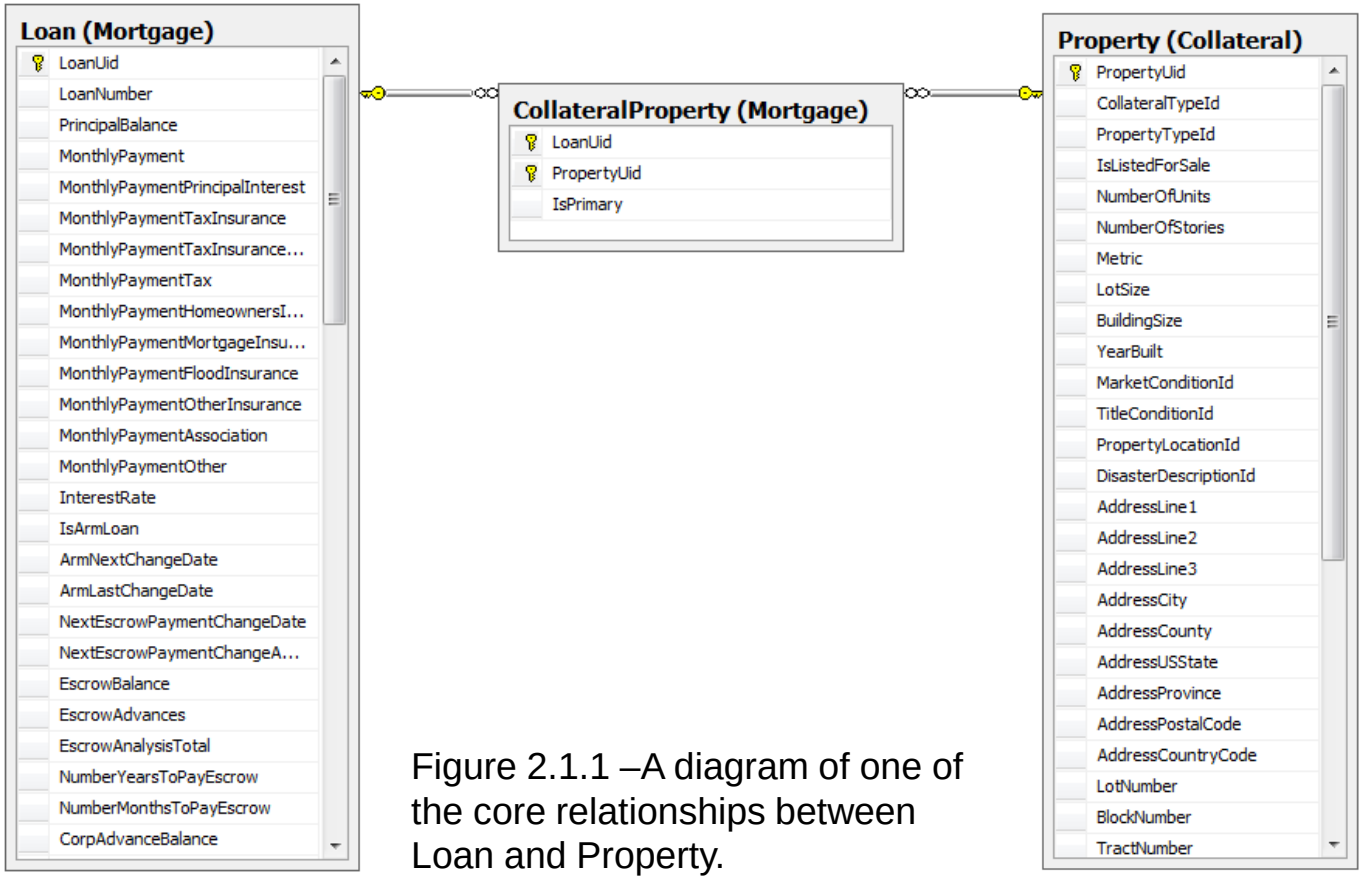


Figure 2.1.1 –A diagram of one of the core relationships between Loan and Property.

Database Tables - Relationships

- We have strived to keep a normalized database that represents real-life business scenarios:
  - **Example #1**:
    - Multiple properties backed by one mortgage.
  - **Example #2**:
    - Multiple mortgages (first and second mortgages) on the same property.
      - NOTE: Although the database is structured to support this scenario, there has yet to be a business request to develop support for it in the UI and Business Object layers.

Database Tables - Standards

- For details on our database standards, see:
  - http://home.servicelinkfnf.com/departments/servicingsolutions/development/Shared%20Documents/Coding%20Standards/Database%20Standards%20for%20Fusion.docx

2.2 MetaData

MetaData

- An XML file that defines a Business Object, its properties, fields and data update/retrieval.

MetaData - Objects

- The framework MetaData bridges the gap between the database and the Business Objects (“objects”).
  - NOTE: The Business Objects are also referred to as “entities” even though we do **not** use Microsoft’s Entity Framework.

MetaData - Example

Figure 2.2.1 –
The contents of a MetaData file collapsed.

```
<MetaData
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="MetaData.xsd"
    Namespace="Dri.Tds.Entities"
    AssemblyName="Dri.Tds.Entities, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
    Version="1.5.0.999">
    <Objects>
        <ObjectInfo
            ObjectName="BorrowerPerson"
            DisplayName="Borrower"
            InheritsFrom="Person"
            ObjectUid="631165f3-a4da-45b9-a891-ecfa5b432b5a"
            EditFormName="FormParts.BorrowerPersonPart"
            Parent="Mortgage"
            DetectUpdateCollision="false">
            <Description>Borrower Person</Description>
            <Fields>...</Fields>
            <Relationships>...</Relationships>
            <DisplayColumnDefinitionSets>...</DisplayColumnDefinitionSets>
            <NavigationItems />
            <Add>...</Add>
            <Read>...</Read>
            <Update>...</Update>
            <Delete />
            <Validations />
            <Rules>...</Rules>
            <Dependencies>...</Dependencies>
        </ObjectInfo>
    </Objects>
</MetaData>
```

MetaData - General

- Each MetaData file is an XML-formatted file that defines aspects of a Business Object including but not limited to:
  - Fields
  - Database create, read, update, delete (“CRUD”) definitions
- Metadata files do not contain calculations.
  - See “Business Objects – Calculations”

MetaData – Generating Objects

- When a MetaData file is saved, our custom tool (called “RinconEntityGenerator”) reads the XML definition and builds the Business Object as a C# Partial Class.
  - See Figure 2.3.1

MetaData - Fields

- A “field” defined in the MetaData adds a typed property on the Business Object (generated partial class) with “get” and “set” accessors.
- Fields usually map directly to Database Table columns.

Figure 2.2.2 – An example of a field in the MetaData.

```
<FieldInfo FieldName="Amount" Label="Amount" EditType="Currency" SystemType="decimal" Scale="4" Precision="19" >
```

MetaData – CRUD Definitions

- “CRUD” information is used by the Data Access Layer to create, read, update and delete from the database.
- Basic CRUD info simply references the Database Table name.

Figure 2.2.3 – An of a basic CRUD definition for creating a new object.

```
<Add>
    <CrudInfo Source="Mortgage.BorrowerIncome" Type="Table"  />
</Add>
```

2.3 Business Objects

Business Object

- A public C# class that gets generated from a MetaData file.

Business Objects - Generated

```
1  using ...
5
6  namespace Dri.Tds.Entities
7  {
8      /// <summary> ...
       25 references
13     public partial class BorrowerIncome : BusinessObject
14     {
15         public const string StaticObjectName = "BorrowerIncome";
16
17         [System.ComponentModel.EditorBrowsable(System.ComponentModel.EditorBrowsableState.Never)]
           6 references
18         public FieldGuid _BorrowerIncomeUid { get; private set; }
19         [System.ComponentModel.EditorBrowsable(System.ComponentModel.EditorBrowsableState.Never)]
           6 references
20         public FieldGuid _FinancialBorrowerUid { get; private set; }
21         [System.ComponentModel.EditorBrowsable(System.ComponentModel.EditorBrowsableState.Never)]
           6 references
22         public FieldInt _IncomeTypeId { get; private set; }
23         [System.ComponentModel.EditorBrowsable(System.ComponentModel.EditorBrowsableState.Never)]
           6 references
24         public FieldDecimal _Amount { get; private set; }
25         [System.ComponentModel.EditorBrowsable(System.ComponentModel.EditorBrowsableState.Never)]
```

Figure 2.3.1: This is how a generated Business Object class looks.

Business Objects - Generated

- A Business Object that is generated from MetaData is a `public partial class` that inherits from a base BusinessObject class.
  - It's essentially a dumb property container.

- Files of generated Business Objects are designated with a ".g.cs" file extension.
  - **Example:** `BorrowerIncome.g.cs`

Business Objects - Calculations

- To add calculations to a Business Object, a new `public partial class` with the same name must be added to the project.
  - This new partial class will be the container for all calculation methods & properties.

- Supplemental partial class files simply end in ".cs" without including the "g".
  - **Example:** `BorrowerIncome.cs`

The End..?

- More to come:
  - DataFields
  - FormParts
  - Forms

- For help with UI, check our "Form Layout Guide"
  - NOTE: it was developed pre-merger and acquisition so the branding isn't official BKFS.