Application Object Tree (AOT)

Lifecycle Services (LCS)

Software Deployment Package (SDP)

(IDE) in Microsoft Dynamics AX is called MorphX.

programmable objects in Microsoft Dynamics AX—tables, forms, reports, classes, and so on—are organized in a tree structure called the Application Object Tree (AOT).

Creating and editing objects is enhanced by drag-and-drop functionality

The source code for methods on classes, tables, forms, and other objects is available to help you extend and customize Microsoft Dynamics AX functionality.

X++ is the programming language in Microsoft Dynamics A

Class Defines an object's interfaces.

instructs or explains how to construct an object of a particular type.

characteristic of a class is that you can create a new instance (an object) of the class.

Forms are an example of a class

Controls A graphical object, such as a text box, a check box, a command button, or a rectangle, that you place on a form or report when you are designing it to display data, perform an acti o n

Data Source Holds the data variables that a form or a query uses

data variables can be o ne or more tables, or they can be individual fields from tables .

Object

Any form or control is an object. The database is an object.

Properties are data that describe an object. Each type of object has different types of properties

A query is a filter mechanism to retrieve the data you want to see from your database tables.

Application object layer

Application object model

**Enums**

Click new item under our USR layer under our current solution in the solution explorer panel

Will then open a window for you what data type to choose which is I this case an enum, and once we specify the type we can then choose what type of enum.

Choose the base enum type and once item is created in our element panel right click the newly created enun and click new element from here we can add the values to our enum and their respective indexes from which each element will have

e.g.

DBFMServiceType

click new element for each of the ff: None, Registration, OilChange, Maintenance, Repair (I assume these are objects that we will create, or maybe not maybe like in C they are just placeholders for numbers/indeces)

To create an enum Expand the Data Dictionary node in the AOT. Right click the Base Enums node and select New Base Enum . Rename the enum. The literals in the enum are called elements. Right-click the enum and select New Element . Rename the element. Add as many additional elements as you need.

Declaring Enums You must create an enum type in the AOT before you can declare it. Enum declaration = enumname Variable { , Variable } ; Variable = identifier [ option ] Option = Arrayoptions | initialization

Lost highlights on this page:

Declaring Enums

You must create an enum type in the AOT before you can declare it.

Enum declaration – enumname Variable { , Variable } ;

Variable – identifier [ option ]

Option – Arrayoptions | initialization

Ex.

//A NoYes enum

NoYes done;

//An array of Criteria enums

Criteria crit[100];

Some more examples of declaring enums in code are:

1. DBFMVehicleType vehicle\_type {Car, Truck, Van, SUV};

DBFMVehicaleType::Car will return 0

1. DBFMFuelType fuel\_type {Gas, Diesel, Propane, Natural Gas, Electric};

enum Django {One, Two, Ps3, Four, Five} stuff; // Variable stuff created to work with

enum Django gamestation = Ps3; // gamestation is now assigned to the Django set, Ps3

gamestation now has a value 2 by default

Best practices:

1. Consider

Length

if it should be mandatory

if it should be a special type

1. Let constant values be enums
2. Never use numeric constants or other constants instead of enums
3. Never use relational operators on enums especially of enums of different types
4. Never assign enums to other enums of different types

**Extended Data Types**

Extended data types (EDTs) are user-defined types, based on the primitive data types boolean, integer, real, string, and date, and the composite type container. You can also base EDTs on other EDTs.

you could create a new EDT called Name and base it on a string

Code is easier to read because variables have a meaningful data type. For example, Name instead of string.

Declaration of EDT Variables In the AOT, the Data Dictionary > Extended Data Types node is used to create EDTs. The range of an EDT is identical to that of the base type it is based on. When you declare a variable in X++, use the syntax shown in the following table.

Extended declaration = Extendedtype Variable { , Variable } ;

Variable = Identifier [ option ]

Option = arrayoptions | initialization

where Extendedtype is the name of the extended data type in the AOT.

X++

// A UserGroupID (integer) variable is declared and initialized to 1.

UserGroupID groupID = 1;

// An Amount (real) variable is declared.

Amount currency;

EmployeeName some\_person = “<some persons name>”

Some EDTs in fleet management scenarios <Type>

* VehicleID or in c terms typedef int VehicleID
* InspectionDate date = <some date value>
* ThirdRowCheckBox trcb1 = <some enum value>
* DBFMGrossVehicleWeight gvw1 = <some int value>

You can create an extended data type in the Application Object Tree (AOT), and then base a table field on this type.

1. In the Application Object Tree, click Data Dictionary
2. Right-click Extended Data Types, click New, and then click a data type to base the extended data type on e.g. string is a the data type the extended data type can base on
3. Right-click the extended data type you created in step 2, and then click Properties
4. To base the extended data type on another extended data type, select an extended data type from the Extends property list

The list of available extended data types varies, depending on the base data type that you selected in step 2 e.g. EDT string, EDT date

1. Modify additional properties, as needed
2. Press CTRL+S to save the extended data type.

* AOT
* Data dictionary

1. Tables
2. Maps
3. Views
4. Extended Data Types
5. MyAddress (USR)
6. Array Elements
7. Address1
8. Address2
9. AdressN
10. Relations
11. Base Enums

Define an Extended Data Type as an Array

1. Expand the relevant extended data type, and then locate the Array Elements node
2. Right-click the Array Elements node, and then choose New Element in the shortcut menu
3. Display the property sheet for the element, and then rename the element by changing the Label property
4. To specify the plural name of the element, use the CollectionLabel property
5. To display a help string when using the extended data type in a form, use the HelpText property
6. Save your modifications.

**Tables, Maps, Views**

Like the Base Enum and Extended Data Types, tables, maps, and views are also elements under the application object tree

Tables, views, and maps are elements that X++ SQL statements can reference to read and write business data

specified in the Application Object Tree (AOT) under AOT > Data Dictionary . The following table describes these elements.

Tables store business data. Each table in the AOT has a corresponding table in the underlying Microsoft SQL Server database.

Method members – A table can have methods, just as a class in X++ or C# can have methods. Table inheritance – A table can extend, or be derived from another table. That same table can be the base table for several derived tables.

My assumption for this is like an entity in Django which is used as basis for creating the table in a data base, a table is also based on an entity, only now we are explicitly creating the entity itself using the table

**Tables**

are the foundation objects in Microsoft Dynamics AX and store data used by the system

made up of records (or rows) that contain information about a single entry in the table.

For example, a specific customer

ecord consists of one or more fields (or columns) that contain a discrete piece of data of a specific data type.

Each table contains the following primary elements:

Objectives in using tables

1. Add data types
2. Extend a pre-built table using inheritance, or by using an already user defined table and inheriting it
3. Set key properties
4. Create a query to create, read, update, patch, and delete an element from the table

Table fields are also based on a primitive data type or an extended data type.

Creating a Table

1. In the AOT expand the Data Dictionary node.
2. Right-click the Tables node, and then select New Table
3. Right-click the table, and then click Properties
4. Rename the table by modifying the Name property
5. To specify the table as temporary, set the Temporary property to Yes. For more information, see Table Properties
6. Modify additional table properties, as needed. For more information, see Table Properties
7. To delete the table, right-click it, and then click Delete

Adding Fields to a Table

Note You can delete only fields that do not contain data in any of the table records. You cannot modify the data type of an existing field

1. Right-click the Fields node of your table
2. Click New and then choose a primitive data type to base your field on. If you plan to base the field on a specific extended data type, you must choose a primitive data type that the extended data type is based on
3. To base the field on an extended data type, set the ExtendedDataType property in the properties panel I guess
4. Modify additional field properties, as needed. For more information
5. To delete the field, right-click it, and then click Delete

Changing table fields during development (like Django we have to find a way to make migrations to all the changes to our entities/tables/models)

When you insert data in a table during development, the SQL statement you use to insert the data is cached in the AOS. Next you might add a new field to the table and persist the change to the database. This causes the SQL statement in the cache to become stale, because the statement is not updated to include the new field. If you reuse the stale statement, the new field is ignored, or an error might occur.

To avoid this problem, restart the AOS after you persist table schema changes to the database. The cache is empty when the AOS restarts.

**fields**

Fields node contains all the fields in the table

Microsoft Dynamics AX performs data validation to ensure that only valid data is entered into each field in the table

Each field in a table has a number of properties that describe the behavior of the field

**Type** property contains the native data type of the field

**ExtendedDataType** property contains the extended data type value (if the field is based on an extended data type

These properties are akin I think to the kwargs we pass when instantiating a model class or the properties of the model class that we create that will be assigned to the necessary fields (in Django)

**Field groups**

Field groups are objects that group together fields that logically belong together, e.g. first\_name and last\_name

Indexes

Full text indexes

**Delete actions**

The DeleteAction element is used to maintain database consistency when a record is deleted. Define delete actions to specify what should occur when data being deleted in the current table is related to data in another table. The delete action values are None, Cascade, Restricted, and Cascade + Restricted.

I assume this is what the on\_delete kwarg does when it is set to models.CASCADE which is essentially as I’ve learned here just a delete action value called cascade. But what is cascade anyway???

**Relations**

Relations define the relationship between two tables that contain related data. Table relations are used to enforce referential integrity among other functions. Table relations are most commonly used in form fields to enable the look-up of information in another table. If a table relation exists, the Lookup button can be used to display a lookup list of values for a particular field

I assume these are akin to the PrimaryKeyField, ForeignKeyField, OneToOneField, ManyToManyField, OneToManyField, in Django

Table relations are most commonly used in form fields to enable the look up of information in another table

If a table relation exists, the lookup button can be used to display a lookup list of values for a particular field

The matching fields typically have the same name in each table.

a SalesOrder table containing orders might have a field called SalespersonID

The Salesperson table, containing the names of sales people, would also have a field called SalespersonID

To create a table relation, specify that the SalesOrder.SalespersonID field is related to the Salesperson.SalespersonID field

Table relationships are created, viewed, and edited in the Application Object Tree (AOT). When a table relation is created in Microsoft Dynamics AX, you must first specify the table involved in the relation and then define the fields in both tables that are related

Conditional relationships can be created by adding a condition to a table relation. Only records that fulfill the condition are included in the relation

**Mappings**

**Methods**

The Methods node displays all the methods available from a table

Add a New Method

1. Browse to the Data Dictionary, Tables node in the AOT.
2. Expand the table, right-click the Methods node, and then select New Method.
3. Enter your code in the Editor window and save your changes.

Override a Method

1. Browse to the Data Dictionary, Tables node in the AOT.
2. Expand the table, right-click the Methods node, and then select Override Method.
3. Select the method that you want to override.
4. Enter your code in the Editor window, and then save your changes.
5. Methods that have been overridden display an icon with an arrow.

Like entities/models in Django that you have created which have the ModelManager class, which have retrieve and create methods we too can override the pre-built methods of a table/entity/model or add a method to according to our needs

**events**

**View**

is an X++ SQL select statement that is given a name that is reusable in other X++ SQL statements

select statement of the view can reference one table, or it can join tables. Also, a view can reference other views, or a mix of views and tables. A view can also reference maps.

A map can unify the access to similar columns and methods that are present in multiple tables

ou associate a map field with a field in one or more tables

enables you to use the same field name to access fields with different names in different tables

Methods on maps enable you to create or modify methods that act on the table fields that the map references

A table name can contain letters and numbers but must begin with a letter. Spaces and special characters are not allowed