Microsoft Dynamics® AX 2012

Developing with Table Inheritance

White Paper

Microsoft Dynamics AX 2012 introduces programming support for table inheritance. This paper outlines the developer experience of creating and programming a table inheritance hierarchy and describes the run-time behavior of various existing and new Microsoft Dynamics AX structures and processes that are involved with table inheritance.

Date: April, 2011

Author: Yanning Liu, Software Development Engineer in Test II

Send suggestions and comments about this document to adocs@microsoft.com. Please include the title with your feedback.

Table of Contents

Introduction

In Microsoft Dynamics® AX 2012, programming models have been enhanced to assist developers in developing applications based on revised or new relational data models. Among these enhancements is the introduction of programming support for the table inheritance data model.

Table inheritance is a concept—facilitated through data modeling—that recognizes and represents generalized and specialized relationships between data entities. In Microsoft Dynamics AX 2012, tables can inherit, or extend, from the tables that are situated above them in a hierarchy. A base table contains fields that are common to all tables that derive from it. A derived table inherits these fields, but also contains fields that are unique to its purpose. Each table contains the **SupportInheritance** and **Extends** properties, which can be used to control table inheritance.

For example, the data model for the global address book (GAB) has been refactored to reduce data redundancy and the number of synchronizations among the master data tables. An essential part of the revised GAB data model is the "party inheritance hierarchy," which holds the master data for business entities (parties). The DirPartyTable table stores the common properties of organizations and people and serves as the root of the hierarchy. The DirOrganizationBase and DirPerson tables inherit from the DirPartyTable table, but they also contain data columns that are specific to organizations and persons, respectively.

Microsoft Dynamics AX 2012 enables developers to enjoy the same object oriented programming support on table inheritance hierarchies as on classes. This support includes field inheritance, table method inheritance, polymorphism, and casting between base and derived table buffers.

This paper outlines the developer experience of creating and programming with the table inheritance data model. In addition, it describes the run-time behavior of various existing and new Microsoft Dynamics AX constructs and processes when table inheritance is involved.

Table inheritance implementation in Microsoft Dynamics AX 2012

Microsoft Dynamics AX 2012 adopts the *table-per-type* storage pattern to support table inheritance data models. In this pattern, each type in the table inheritance hierarchy is stored in a separate table in the back-end database. This approach avoids creating wide master data tables, and thereby reduces storage demand, which is crucial in boosting data upgrade performance. It also simplifies relational modeling, because the Microsoft Dynamics AX 2012 data access stack recognizes the relations defined across the table inheritance hierarchy and manages the stored data accordingly.

The Microsoft Dynamics AX 2012 implementation of table inheritance imposes the disjoint rule for the disjointness constraint on the table inheritance data models that it supports. The disjoint rule requires that an instance of the base type can only be one of the derived types. In addition, converting a table inheritance record to a different type at run time is not allowed in Microsoft Dynamics AX 2012.

Data access on table inheritance hierarchies is completely managed by Microsoft Dynamics AX 2012 to allow developers to perform the **insert**, **update**, and **delete** operations on the table inheritance hierarchy in the same way that they do on tables that are not involved in inheritance hierarchies. A query from an inheritance table is polymorphic and the returned data will include the records of the derived types.

Performance considerations in table inheritance

The modeling and programming benefits of table inheritance come at a price: a certain degree of performance degradation is incurred when data access is performed on table inheritance hierarchies. This degradation is associated with the extra costs of navigating and persisting (or joining) data on multiple tables in the hierarchy, as compared to working with a single table when no table inheritance is involved. Developers are advised to evaluate the performance implications when implementing their solutions by using table inheritance data models.

As a general guideline, the table inheritance data model should be avoided on transactional tables to minimize the performance impact. In addition, developers are advised to avoid creating extensively deep and wide table inheritance hierarchies when designing the physical data models; this will sometimes require collapsing the hierarchy. When querying data from an inheritance hierarchy, developers should design the query from the *concrete* table types (explained in the Abstract vs. concrete tables section). In addition, to reduce the extra performance costs associated with polymorphism, they should use polymorphic queries only when necessary.

Microsoft Dynamics AX 2012 has enhanced the ad hoc query mode to alleviate the back-end database load that could result from the large number of table joins when querying data from table inheritance hierarchies. Developers are encouraged to use the ad hoc query mode when only partial data is needed.

Terminology

Microsoft Dynamics AX 2012 terms:

Term	Definition
Root table	A table at the head of the table inheritance hierarchy.
Leaf table	A table at the bottom of the table inheritance hierarchy; no other tables inherit from this table.
Base table	A table with characteristics that can be inherited by other tables. A table that functions as a base table can also be a derived table if it inherits some of its characteristics from tables further up in the hierarchy.
Derived table	A table that inherits some of its characteristics from one or more base tables.
Type	A unique data entity in a data model, represented by a table. The fields of a table can be joined with the fields from the base tables (if any) in a table buffer to represent all the fields for the type.

Implementing a table inheritance hierarchy in the AOT

This section explains how to construct a table inheritance hierarchy in the Application Object Tree (AOT) and how to configure the properties that govern the run-time behavior of the hierarchy.

Designing a table inheritance hierarchy

The *table-per-type* representation of the table inheritance data model in Microsoft Dynamics AX 2012 provides developers with the convenience of overloading the Tables node of the AOT to define the table inheritance entities.

The process of creating the table inheritance hierarchy starts with creating the data tables that map to the table inheritance entities in the AOT. It is important to note that Microsoft Dynamics AX 2012 does not allow a table to join a table hierarchy if its fields have already been defined. This means that the tables you create, which will represent the table inheritance entities, must not contain any fields before the inheritance relationships among them have been defined in the AOT.

Developers should also be aware that, after the inheritance relationships have been defined, certain table properties become disabled on the table inheritance tables, with the exception of the root table. This behavior applies to properties that belong to the inheritance hierarchy rather than to the individual table inheritance type.

However, a table inheritance hierarchy can only be defined on a normal table type. Microsoft Dynamics AX 2012 does not support creating table inheritance on either InMemory or TempDB table types.

A type discriminator field must be defined on any table inheritance hierarchy created in the AOT. The field must be defined as an **int64** type on the root table, with the name of the field set to

InstanceRelationType. Furthermore, the developer must select this field as the value for the **InstanceRelationType** table property on the root table. If these requirements are not met, a compilation error will occur when the table inheritance hierarchy is compiled.

The InstanceRelationType field of the root table is read-only and stores the TableIDs of record instances; it is populated automatically by Microsoft Dynamics AX 2012.

Constructing a table inheritance hierarchy

For demonstration purposes, we will create a table inheritance hierarchy based on a mock party data model represented by the Entity-Relationship (ER) diagram in Figure 1.

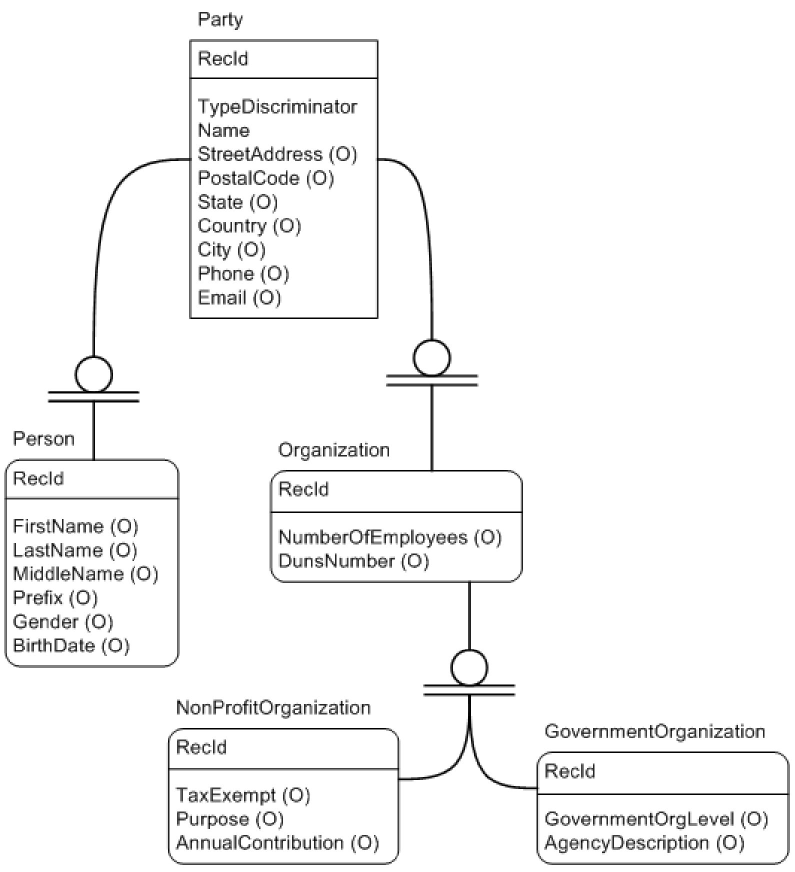


Figure 1: Mock party data model

To create the table inheritance hierarchy

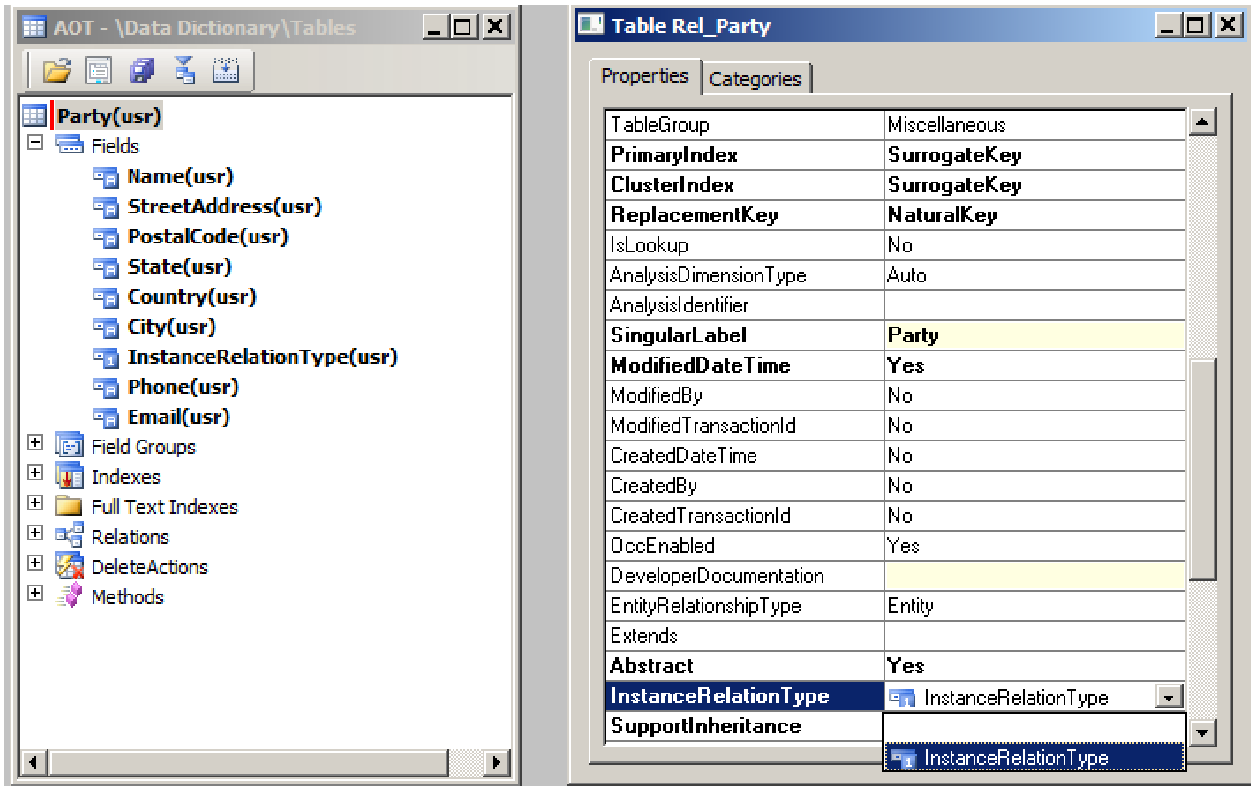
1. Launch the Microsoft Dynamics AX 2012 client in the development workspace.
2. In the AOT, navigate to **Data Dictionary** > **Tables**, and right-click the **Tables** node.
3. Click **New Table**. In the table property window, set the **Name** property value to “Party” and set the **SupportInheritance** property to **Yes**. Do not create any field in the table. Save the table.

   Note: Table inheritance properties such as **Extends** become enabled only if the **SupportInheritance** property has been set to **Yes**.

4. Repeat step 3 to create each of the other tables (Person, Organization, NonProfitOrganization, and GovernmentOrganization) in the mock party data model, setting the **Name** property value to the appropriate name each time.
5. Define the inheritance relationships on the tables as follows:
    a. Select the Person table in the table property window, expand the drop-down list in the **Extends** property and select **Party**. Save the table.
    b. Select the Organization table in the table property window, expand the drop-down list in the **Extends** property, and select **Party**. Save the table.
    c. Select the NonProfitOrganization table in the table property window, expand the drop-down list in the **Extends** property, and select **Organization**. Save the table.
    d. Select the GovernmentOrganization table in the table property window, expand the drop-down list in the **Extends** property, and select **Organization**. Save the table.
    e. Leave the **Extends** property on the base table Party blank.
6. Create table fields according to the date model on the tables.

    Note: Field names must be unique across the entire table inheritance hierarchy. This requirement ensures that the field name uniquely identifies the field across the hierarchy when appearing in system APIs that reference table fields. Developers will also notice that field IDs are unique across the entire table inheritance hierarchy.

7. Configure the type discriminator as follows:
    a. Select the base table Party.
    b. Create an **int64** field and name it **InstanceRelationType**.
    c. In the table property window, expand the drop-down list of the **InstanceRelationType** property, and select **InstanceRelationType**, as shown in the following illustration.


AOT - \Data Dictionary\Tables

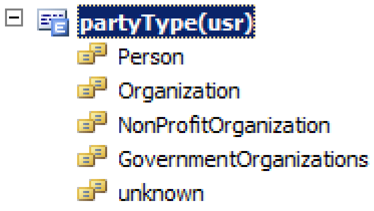
- Party(usr)
  - Fields
    - Name(usr)
    - StreetAddress(usr)
    - PostalCode(usr)
    - State(usr)
    - Country(usr)
    - City(usr)
    - InstanceRelationType(usr)
    - Phone(usr)
    - Email(usr)
  - Field Groups
  - Indexes
  - Full Text Indexes
  - Relations
  - DeleteActions
  - Methods

Table Rel_Party

TableGroup	Miscellaneous
PrimaryIndex	**SurrogateKey**
ClusterIndex	**SurrogateKey**
ReplacementKey	**NaturalKey**
IsLookup	No
AnalysisDimensionType	Auto
AnalysisIdentifier	
SingularLabel	**Party**
ModifiedDateTime	**Yes**
ModifiedBy	No
ModifiedTransactionId	No
CreatedDateTime	No
CreatedBy	No
CreatedTransactionId	No
OccEnabled	Yes
DeveloperDocumentation	
EntityRelationshipType	Entity
Extends	
Abstract	**Yes**
InstanceRelationType	InstanceRelationType
SupportInheritance	

8. Optional: Complete this step only if you are displaying the instance type of the record on a form, and a report is needed.

   Create a **partyType** enumeration type and define the elements, as shown in the following illustration.


- partyType(usr)
  - Person
  - Organization
  - NonProfitOrganization
  - GovernmentOrganizations
  - unknown

   Define an enumeration field of **partyType** on the Party base table. Name the field **DisplayRelationType**. Override the **insert** table method on the Party table.

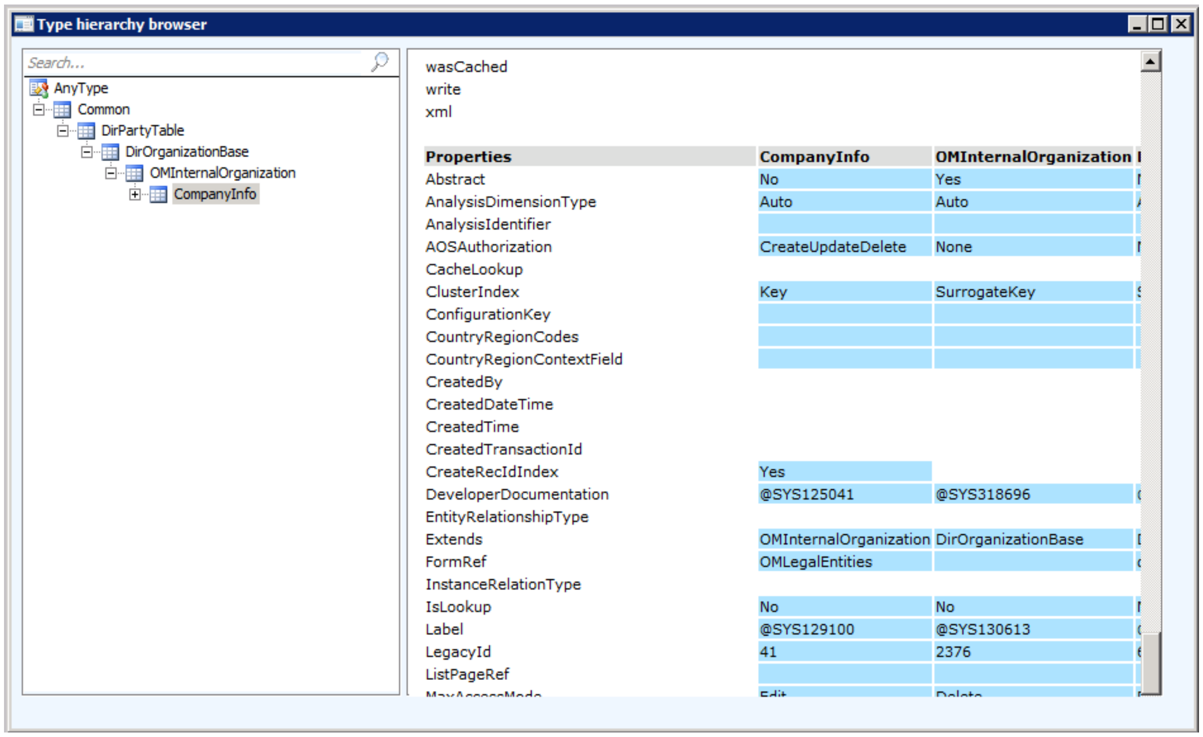
```
public void insert()
{
      DictEnum enumType = new DictEnum(enumnum(partytype));
      int enumValue;
      if (this.DisplayRelationType != partytype::Unknown)
      {
              return;
      }
      enumValue = enumType..symbol2Value(this.getInstanceRelationType());
      if (enumValue == 255) // symbol2Value returns 255 if cannot convert
      {
```

```
                throw error(strfmt('No %1 enum element found for EnumValue %2',
                            enumstr(partytype),
                           this.getInstanceRelationType()));
            }
            else
            {
                this.DisplayRelationType = enumValue;
            }
            super();
      }
```

9. Save and compile the tables in the AOT. Make sure there is no compilation error.

Completing these steps effectively "translates" the data model to the AOT. Developers can also expand the hierarchy. This is accomplished by connecting additional tables to the hierarchy through setting the proper value of the **Extends** property. (Correspondingly, setting the **Extends** table property value to "empty" on a derived table will disconnect the table from the hierarchy.) This must be done with caution, however, because Microsoft Dynamics AX 2012 only allows tables with an empty field list to join the inheritance hierarchy.

Developers can now use a new developer tool, the *Type hierarchy browser* (see Figure 2)**,** to visualize the table inheritance hierarchy in a more intuitive way. Developers can access the tool by right clicking the AOT **Tables** node, and then clicking **Add Ins** > **Type hierarchy browser**.



wasCached
write
xml

Properties	CompanyInfo	OMInternalOrganization
Abstract	No	Yes
AnalysisDimensionType	Auto	Auto
AnalysisIdentifier		
AOSAuthorization	CreateUpdateDelete	None
CacheLookup		
ClusterIndex	Key	SurrogateKey
ConfigurationKey		
CountryRegionCodes		
CountryRegionContextField		
CreatedBy		
CreatedDateTime		
CreatedTime		
CreatedTransactionId		
CreateRecIdIndex	Yes	
DeveloperDocumentation	@SYS125041	@SYS318696
EntityRelationshipType		
Extends	OMInternalOrganization	DirOrganizationBase
FormRef	OMLegalEntities	
InstanceRelationType		
IsLookup	No	No
Label	@SYS129100	@SYS130613
LegacyId	41	2376
ListPageRef		
MaxAccessMode	Edit	Delete

Figure 2: Type hierarchy browser

Configuring table inheritance hierarchies in the AOT

After you have constructed the table inheritance hierarchy, you can begin to specify its characteristics, based on the data model from which it originated. Several configuration elements influence the development and definition of a table inheritance hierarchy in the AOT. Some key elements include:

- Abstract versus concrete tables
- Hierarchy versus table properties
- Table methods
- Relations on the table inheritance hierarchy
- Configuration keys on the table relation hierarchy
- Enhanced reflection APIs and scope

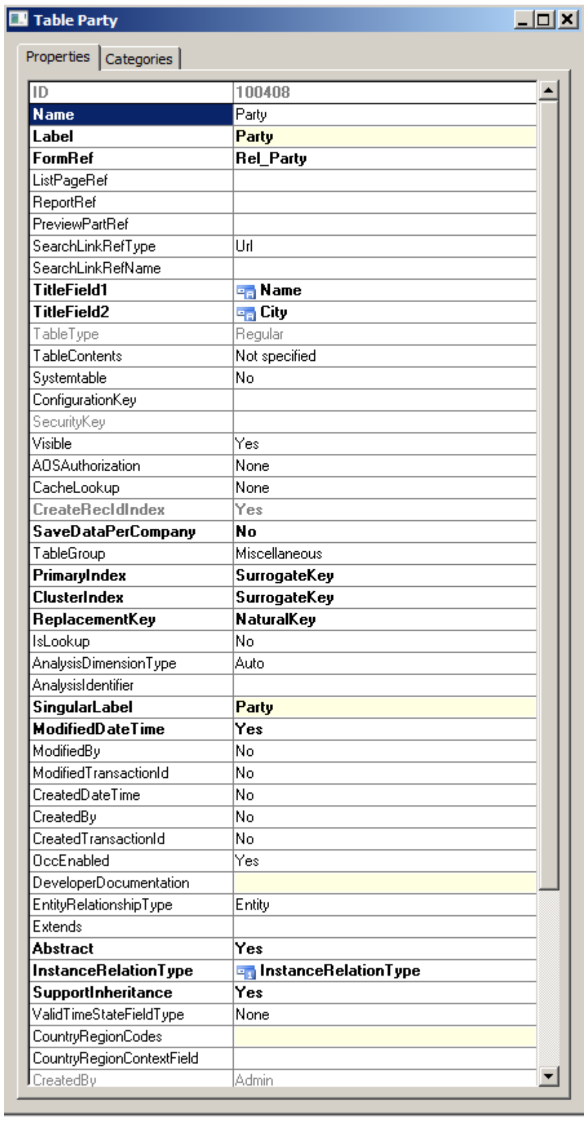
Abstract versus concrete tables

Tables in a table inheritance hierarchy can be defined as either *abstract* or *concrete*, depending on whether the table property **Abstract** is set to **Yes** or **No**. Records can only be created for concrete table types. Any attempt to create a record and insert it in an abstract table will result in a run-time error. The position of the table in the inheritance hierarchy does not restrict its ability to be defined as abstract.

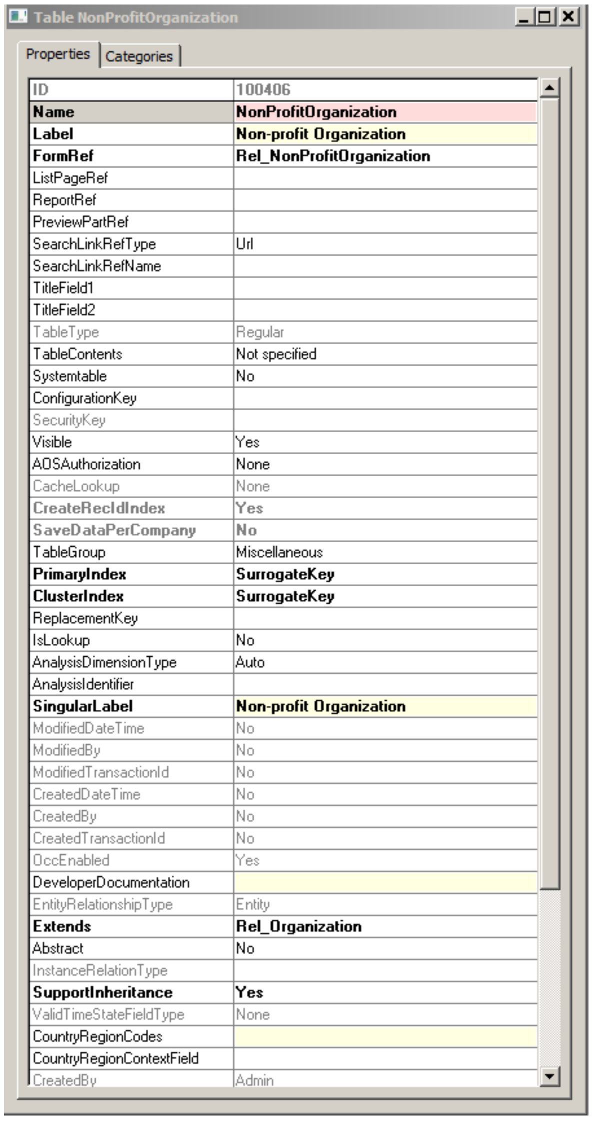
Hierarchy properties versus table properties

Certain properties and run-time behaviors can only be defined on the root table of the table inheritance hierarchy. These properties are considered hierarchy *properties* and their values govern the behaviors of the inheritance tables across the entire hierarchy. Figure 3 captures the table properties of the Party base table and one of its derived tables, NonProfitOrganization, in the mock party data model. Some of the hierarchy properties (which are unavailable in the NonProfitOrganization property list) include:

- **SaveDataPerCompany**
- **CacheLookup**
- **ModifiedDateTime**
- **ModifiedBy**
- **ModifiedTransactionId**
- **CreatedDateTime**
- **CreatedBy**
- **CreatedTransactionId**
- **OccEnabled**
- **EntityRelationshipType**
- **InstanceRelationType**
- **ValidTimeStateFieldType**



ID	100408
Name	Party
Label	Party
FormRef	Rel_Party
ListPageRef	
ReportRef	
PreviewPartRef	
SearchLinkRefType	Url
SearchLinkRefName	
TitleField1	Name
TitleField2	City
TableType	Regular
TableContents	Not specified
Systemtable	No
ConfigurationKey	
SecurityKey	
Visible	Yes
AOSAuthorization	None
CacheLookup	None
CreateRecIdIndex	Yes
SaveDataPerCompany	No
TableGroup	Miscellaneous
PrimaryIndex	SurrogateKey
ClusterIndex	SurrogateKey
ReplacementKey	NaturalKey
IsLookup	No
AnalysisDimensionType	Auto
AnalysisIdentifier	
SingularLabel	Party
ModifiedDateTime	Yes
ModifiedBy	No
ModifiedTransactionId	No
CreatedDateTime	No
CreatedBy	No
CreatedTransactionId	No
OccEnabled	Yes
DeveloperDocumentation	
EntityRelationshipType	Entity
Extends	
Abstract	Yes
InstanceRelationType	InstanceRelationType
SupportInheritance	Yes
ValidTimeStateFieldType	None
CountryRegionCodes	
CountryRegionContextField	
CreatedBy	Admin



ID	100406
Name	NonProfitOrganization
Label	Non-profit Organization
FormRef	Rel_NonProfitOrganization
ListPageRef	
ReportRef	
PreviewPartRef	
SearchLinkRefType	Url
SearchLinkRefName	
TitleField1	
TitleField2	
TableType	Regular
TableContents	Not specified
Systemtable	No
ConfigurationKey	
SecurityKey	
Visible	Yes
AOSAuthorization	None
CacheLookup	None
CreateRecIdIndex	Yes
SaveDataPerCompany	No
TableGroup	Miscellaneous
PrimaryIndex	SurrogateKey
ClusterIndex	SurrogateKey
ReplacementKey	
IsLookup	No
AnalysisDimensionType	Auto
AnalysisIdentifier	
SingularLabel	Non-profit Organization
ModifiedDateTime	No
ModifiedBy	No
ModifiedTransactionId	No
CreatedDateTime	No
CreatedBy	No
CreatedTransactionId	No
OccEnabled	Yes
DeveloperDocumentation	
EntityRelationshipType	Entity
Extends	Rel_Organization
Abstract	No
InstanceRelationType	
SupportInheritance	Yes
ValidTimeStateFieldType	None
CountryRegionCodes	
CountryRegionContextField	
CreatedBy	Admin

Figure 3: Hierarchy and table properties

Table properties not listed above are regular table properties that are not overloaded for defining hierarchy behaviors. The properties **TitleField1** and **TitleField2**, however, will inherit the values from the base table if not defined on the current table. This inheritance principle also applies to the **Field Group** definitions (such as **AutoIdentification**, **AutoLookup**, **AutoReport** and so on) under a table node on table inheritance hierarchies.

Table methods

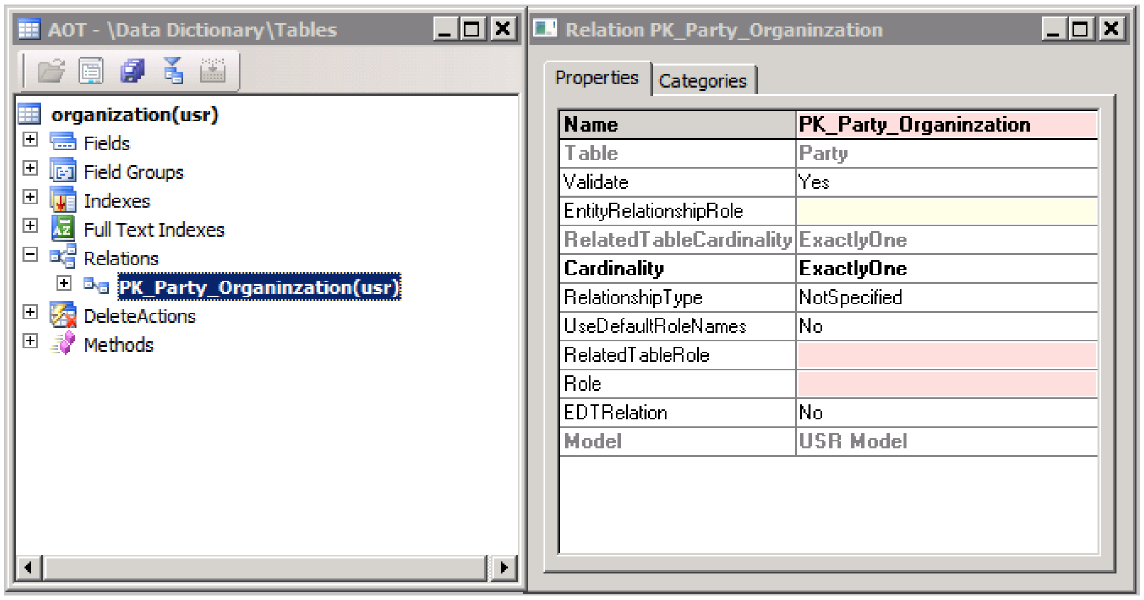
Microsoft Dynamics AX 2012 provides the same support for table method inheritance and polymorphism on the table inheritance hierarchy that developers encounter with classes. Developers can access the base table method on a derived table buffer. In addition, a derived table method can override a base table method and call **super**() to invoke the same method on the base table buffer. Upcasting and downcasting among base table and derived tables instances is also supported. The actual methods invoked depend on the instance type at run time.

Certain limitations apply, as follows:

- There is no support for tables to implement interfaces.
- There is no support for specifying that a table is final (and therefore not subject to being overridden).
- If an incompatible method signature consisting of both the parameter types and the return type is specified on an overridden method (where the method name remains the same), the compiler will issue an error.

Relations on the table inheritance hierarchy

Microsoft Dynamics AX 2012 automatically creates relations between the base table and its derived tables. Naming these relations follows the convention *PK_Base table_Derived table*, which is the data storage pattern that Microsoft Dynamics AX 2012 has adopted for table inheritance hierarchies (see Figure 4).



Name	PK_Party_Organinzation
Table	Party
Validate	Yes
EntityRelationshipRole	
RelatedTableCardinality	ExactlyOne
Cardinality	ExactlyOne
RelationshipType	NotSpecified
UseDefaultRoleNames	No
RelatedTableRole	
Role	
EDTRelation	No
Model	USR Model

Figure 4: Automatically created relations between the base and derived tables

Developers can define extra relations on tables involved in the table inheritance hierarchy that are in keeping with the data model. These relations must be named uniquely across the entire hierarchy because they will be visible to the base and derived tables when the hierarchy is involved in designing queries or forms.

Configuration keys on the table inheritance hierarchy

Configuration keys are often arranged in hierarchies of functionality, mirroring the hierarchies of functionality in the application. Developers can assign a configuration key at any level in the table inheritance hierarchy. However, they must make sure that when a configuration key is assigned to a given table in the inheritance hierarchy, all of its derived tables are assigned either the same configuration key or one of its children keys in the configuration key hierarchy. Not assigning a configuration key will result in the default configuration key behavior in the system, which makes that table available during a basic installation.

A compatible configuration key setting on the table inheritance hierarchy warrants that when a table is disabled (by disabling its assigned configuration key); the entire derived branch of the table is disabled. Any data access activity on a table disabled by configuration key will render no action from the system.

A configuration key compatibility check is enforced at compilation; failing the compatibility check will result in compilation errors.

Enhanced dictionary reflection APIs and scope

In Microsoft Dynamics AX 2012, the dictionary reflection APIs have been enhanced to accommodate the need to discover and traverse the hierarchy to find the needed properties to reflect. A system enumeration type, **TableScope,** which is composed of three values—**CurrentTableOnly, IncludeBaseTables** and **IncludeDerivedTables**—has been introduced to define the scope of the search. An optional parameter of enumeration type **TableScope** is added to the relevant dictionary reflection APIs to specify the scope of reflection discovery. The following is a list of relevant APIs to reflect the table inheritance hierarchy:

- **DictTable.extends**
- **DictTable.extendedBy**
- **DictTable.fieldCnt**
- **DictTable.fieldCnt2Id**
- **DictTable.fieldNext**
- **DictTable.fieldGroupCnt**
- **DictTable.fieldGroup**
- **DictTable.relation**
- **DictTable.relationCnt**
- **DictTable.objectMethodCnt**
- **DictTable.objectMethod**
- **DictTable.objectMethodObject**
- **DictTable.titleField1**
- **DictTable.titleField2**
- **DictRelation.loadTableRelation**
- **DictRelation.loadFieldRelation**

Implicit enhancements to a range of reflection APIs were also made to supply these reflection APIs or intrinsic functions with fields or relations defined uniquely on the inheritance hierarchy. In other words, you do not need to specify a table scope for these reflection APIs. Examples are as follows:

- **DictTable.fieldName**
- **DictTable.fieldObject**
- **DictTable.fieldName2Id**
- **DictTable.fieldNum**

Programming table inheritance

A new set of programming artifacts has been introduced in Microsoft Dynamics AX 2012. These artifacts provide transparent, end-to-end support for programming with table inheritance data models and run-time support for accessing data on table inheritance hierarchies.

- **Field inheritance –** The table buffer of a derived table in an inheritance hierarchy can directly access the fields defined on the base tables.

- **Table buffer casting –** Direct upcasting is allowed between base and derived table record instances, and downcasting is allowed with the help of **is** and **as** operators.
- **Table method inheritance –** The table buffer of a derived table is able to directly call methods defined on the base tables.
- **Table method polymorphism –** The table method invoked on a table buffer is determined by the instance type of the record in the buffer at run time.

Inserting data into a table inheritance hierarchy

The following code example demonstrates how to insert records into a concrete type table—in this case, the GovernmentOrganization table in the mock party data model.

```
static void Job_InsertGovOrg(Args _args)
{
     GovernmentOrganization tstGov;
    NonProfitOrganization  tstNpo;

    tstNpo.Name  = "Jaguar Concert Hall";
    tstNpo.Email = "email@JaguarConcert.Org";
    tstNpo.State = "IL";
    tstNpo.City  = "Urbana";
    tstNpo.DunsNumber = " JagCont001";
    tstNpo.NumberOfEmployees = 10;
    tstNpo.AnnualContribution = 12345.67;
    tstNPO.insert();

    tstGov.Name  = "Illinois State Tax Authority";
    TstGov.Email = "Tax@il.gov";
    tstGov.State = "IL";
    tstGov.City  = "Springfield";
    tstGov.DunsNumber = "ILTAX001";
    tstGov.NumberOfEmployees = 200;
    tstGov.AgencyDescription = "Illinois State Tax Authority";
    tstGov.insert();

    info(tstGov.getInstanceRelationType()); //GovermentOrganization
}
```

Microsoft Dynamics AX 2012 manages committing the data in a transaction across the relevant tables in the inheritance hierarchy at run time when an insert operation is issued on a concrete type table buffer. The inheritance relationship is reflected in the data by the propagating surrogate keys. The data persisted in the database tables will look as follows.

GovermentOrganization table data

AgencyDescription	RecId
Illinois State Tax Authority	5637145080

NonprofitOrganization table data

Annual Contribution	RecId
12345.67	5637145079

Organization table data

Number of Employees	DUNS Number	RecId
10	JagCont001	5637145079
200	ILTAX001	5637145080

Party table data

Name	State	City	InstanceRelationType	Email	RecId
Jaguar Concert Hall	IL	Urbana	100433	email@JaguarConcert.Org	5637145079
Illinois State Tax Authority	IL	Springfield	100434	Tax@il.gov	5637145080

The InstanceRelationType field defined on the Party base table is populated by Microsoft Dynamics AX 2012 and stores the TableIDs of the concrete types. However, when the type of a record instance is needed, the developer should use the **common.getInstanceRelationType** API instead of directly accessing the InstanceRelationType field.

Updating data on the table inheritance hierarchy

The following code example demonstrates how to update records in the table inheritance hierarchy.

Note that the update can be issued from either the concrete (derived) type table buffer or the base type table buffer.

```
static void UpdateOrg(Args _args)
{
    Organization tstOrg;
    NonProfitOrganization tstNpo;

    ttsBegin;
        while select forupdate * from tstOrg
        {
            tstOrg.State = "IL";
            tstOrg.NumberOfEmployees = tstOrg.NumberOfEmployees+10;
            tstOrg.update();
        }
       select forUpdate * from tstNpo;
       tstNpo.AnnualContribution = 76543.21;
       tstNpo.update();
    ttsCommit;
 }
```

Running the UpdateOrg code will produce the following data in the tables.

GovermentOrganization table data

AgencyDescription	RecId
Illinois State Tax Authority	5637145080

NonprofitOrganization table data

Annual Contribution	RecId
76543.21	5637145079

Organization table data

Number of Employees	DUNS Number	RecId
20	JagCont001	5637145079
210	ILTAX001	5637145080

Party table data

Name	State	City	InstanceRelationType	Email	RecId
Jaguar Concert Hall	IL	Urbana	100433	email@JaguarConcert.Org	5637145079
Illinois State Tax Authority	IL	Springfield	100434	Tax@il.gov	5637145080

It is important to note that *optimistic concurrency control* (OCC) is applied to the concrete type buffer. Therefore, the application runtime will detect the update conflict if multiple processes attempt to update the same concrete type record, even if the updates occur on different back-end database tables.

Deleting data from the table inheritance hierarchy

As with the **update** method, calling the **delete** method on a table inheritance table buffer deletes the data from the back-end database tables in which the records to be deleted are stored.

Note: When a delete is issued from a base type table buffer, it will also delete the records of the derived types of the base type. The following code example demonstrates this action. After running the **DeleteOrg** code, the Organization, NonProfitOrganization, and GovernmentOrganization tables will all be empty. The Party table does not contain any records of Organization and its derived types, and therefore is not affected.

```
static void DeleteOrg(Args _args)
{
    Organization tstOrg;
    ttsBegin;
        while select forupdate * from tstOrg
        {
            tstOrg.delete();
        }
   ttsCommit;
 }
```

Invoking table methods and table buffer casting

The following code examples demonstrate table method inheritance and polymorphism. It is assumed that the developer has defined the table method **JustSayHello** on the Party, Organization, and the GovernmentOrganization tables.

Code example: Defining the JustSayHello method on multiple tables

```
public void JustSayHello()     //GovernmentOrganization
{
    info("Hello GovernmentOrganization");
    super();
}

public void JustSayHello()     //Organization
{
    info("Hello Organization");
    super();
}

public void JustSayHello()     //Party
{
    info("Hello Party");
}
```

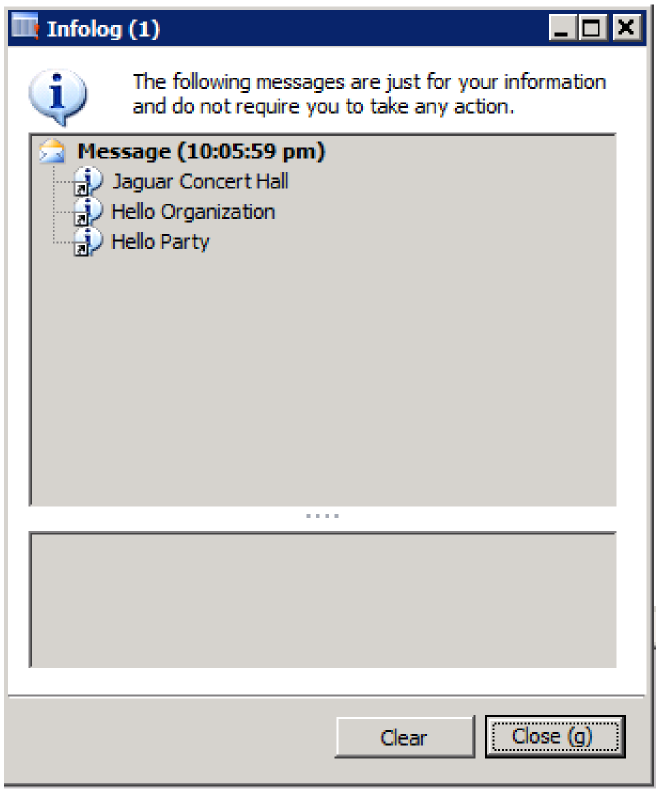
The **super**() keyword instructs the runtime to invoke the overridden method on the nearest base table. It is important to include the call to **super**() in case the table trigger methods (**insert**, **update**, and **delete**) are to be overridden. This is because calls to **super**() inside the trigger methods on the root table would invoke the kernel implementation of data manipulation operations.

Code example: Table method inheritance

The TableMethodInheritance code example demonstrates table method inheritance. You can invoke methods derived from a base table directly from the derived table buffer.

```
static void TableMethodInheritance(Args _args)
{
NonProfitOrganization tstNpo;

select firstOnly tstNpo;
info(tstNpo.Name);
tstNpo.JustSayHello();

}
```

Running the TableMethodInheritance code will produce the results shown in Figure 5. Because we do not override the **JustSayHello** method on the NonProfitOrganization table but only on the Organization and Party, calling **JustSayHello** on the NonProfitOrganization table buffer invokes the **JustSayHello** method defined on Organization and Party sequentially, as chained upwards by the **super**() calls.



The following messages are just for your information and do not require you to take any action.

```
Message (10:05:59 pm)
    Jaguar Concert Hall
    Hello Organization
    Hello Party
```

Figure 5: Results of running TableMethodInheritance code

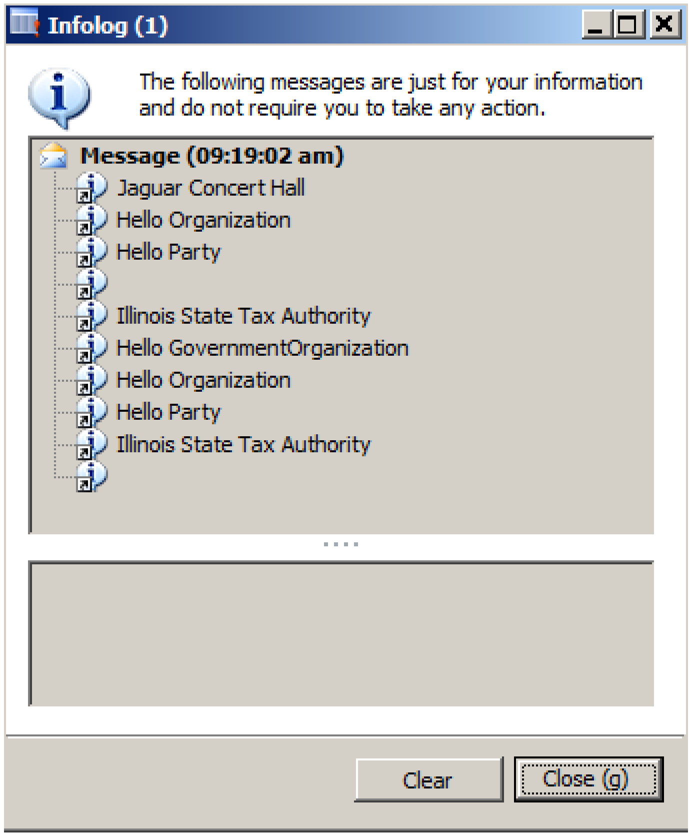
Code example: Table method polymorphism

The PolymorphismAndDownCast code demonstrates polymorphism when table methods are invoked from the Party table.

```
static void PolymorphismAndDownCast(Args _args)
{
    Party                  tstParty;
    GovernmentOrganization tstGov;

    while select tstParty
    {
        info(tstParty.name);
        tstParty.JustSayHello();             //polymorphism
        if(tstParty is GovernmentOrganization)                         //downCast
        {
           tstGov = tstParty as GovernmentOrganization;
           info(tstGov.AgencyDescription);
        }
        info(" ");
    }
}
```

The results of running the code are displayed in Figure 6. It shows that the NonProfitOrganization **JustSayHello** method and GovernmentOrganization **JustSayHello** method are invoked from the **while select** loop, as is demonstrated by the type of record instances retrieved.

Downcasting is necessary when you need to access the fields defined on derived types. The code sample also demonstrates the preferred way of downcasting from a base type table buffer to the derived type table buffer using the **is** and **as** operators.



The following messages are just for your information and do not require you to take any action.

```
Message (09:19:02 am)
  Jaguar Concert Hall
  Hello Organization
  Hello Party

  Illinois State Tax Authority
  Hello GovernmentOrganization
  Hello Organization
  Hello Party
  Illinois State Tax Authority
```

Figure 6: Results of running PolymorphismAndDownCast code

Querying data from a table inheritance hierarchy

X++ select statement

Field inheritance enables the developer to use any fields from concrete and base tables when composing X++ **select** statements. The entire set of fields is accessed from the table buffer that is returned from the query.

The following code example demonstrates querying customers that are of type NonProfitOrganization in the mock party model by using the X++ **select** statement.

```
static void SelectCustomer(Args _args)
{

    NonProfitOrganization npo;
    Customer          cust;
    while select firstonly * from npo join cust
                         order by npo.Name
                         where
                            Cust.party == npo.recid
                         && npo.NumberOfEmployees>100
                         && npo.state == 'IL'
    {
        info(cust.AccountNumber);
        info(npo.Name);
    }
}
```

The Customer table schema has a foreign key relationship to the Party table, as shown in Figure 7.

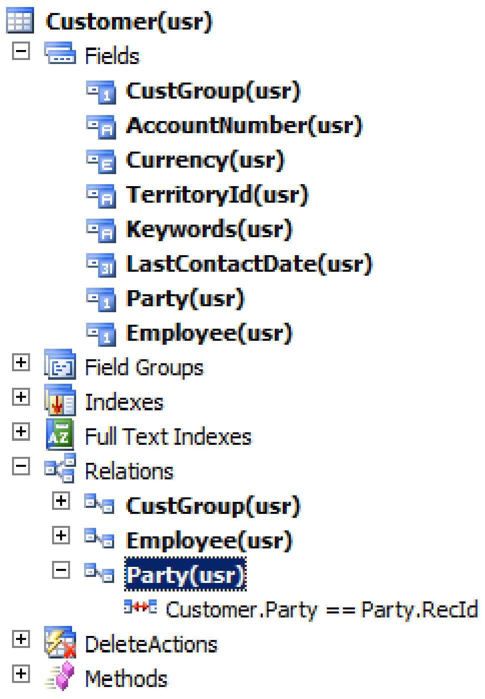

```
Customer(usr)
  Fields
      CustGroup(usr)
      AccountNumber(usr)
      Currency(usr)
      TerritoryId(usr)
      Keywords(usr)
      LastContactDate(usr)
      Party(usr)
      Employee(usr)
  Field Groups
  Indexes
  Full Text Indexes
  Relations
    CustGroup(usr)
    Employee(usr)
    Party(usr)
        Customer.Party == Party.RecId
  DeleteActions
  Methods
```

Figure 7: Example of a foreign key relationship

Support from the query framework

The query framework in Microsoft Dynamics AX 2012 has been enhanced to support constructing query objects from inheritance tables. Developers can construct data source field lists, query ranges, relations, "Group By" and "Order By" conditions, and the newly introduced "Having" condition with inherited fields and fields defined on derived types, when the data sources for the query are inheritance tables.

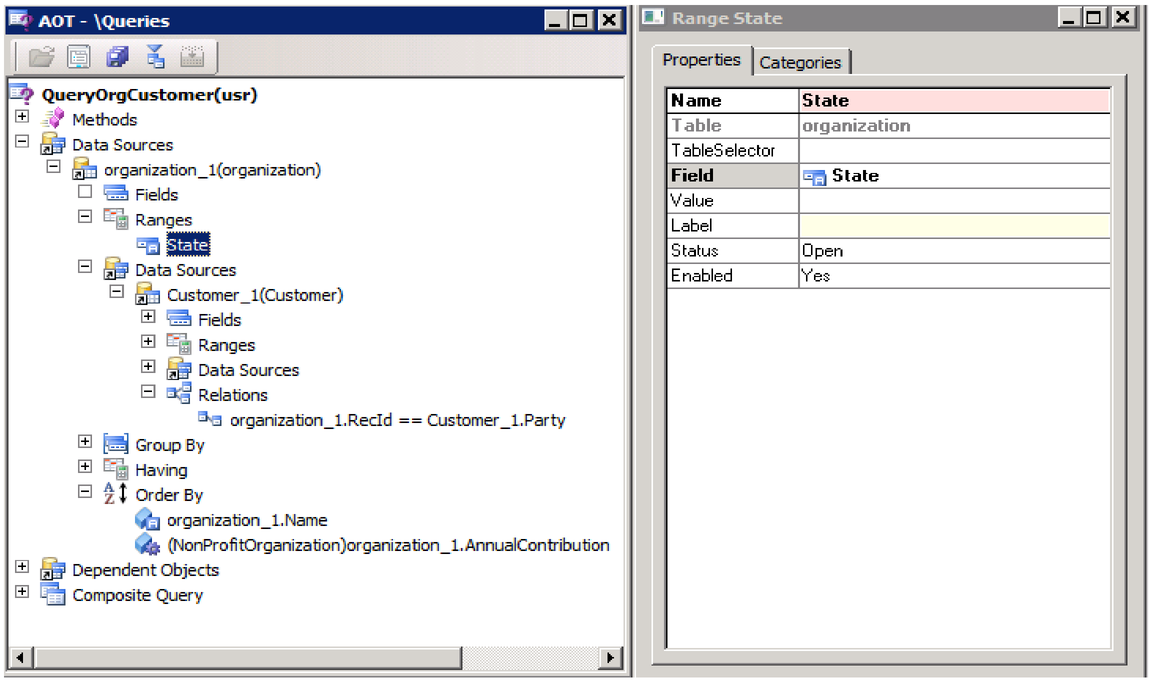
This support is demonstrated by creating an AOT query object that queries the Organization and Customer records as specified by the following **select** statement:

```
select * from Organization join customer
                        order by Organization.Name,
organization(nonProfitOrganization.AnnualContribution)
       where
                             Customer.party == Organization.recid
                          && Organization.state == 'IL'
```

To create the query

1. In the AOT, right-click **Queries**, and then click **New Query**.
2. Right-click the **Data Sources** node and then click **New Data Source**. Specify the Organization table as the data source table.
3. Right-click **Data Sources** under the Organization **Data Sources** node, and then click **New Data Source**. Specify the Customer table as the data source table.
4. Set the relation between the Organization and Customer data sources as follows:
    a. Right-click the **Relations** node under the **Customer** data source, and then click **New Relation**.
    b. In the property window, select **RecId** for the **Field** property and **Party** for the **RelatedField** property.
5. Create a query "Range" condition on the inherited field as follows:
    a. Right-click the **Ranges** node under the Organization **Data Sources** node, then click **New Range.**
    b. In the property window, select the inherited **State** field for the **Field** property and enter **IL** for the **Value** property.
6. Create an "Order By" condition on the inherited **Name** field as follows:
    a. Right-click the **Order By** node and then click **New Field**.
    b. In the property window, select the inherited **Name** field for the **Field** property.
7. Create an "Order By" condition on the AnnualContribution field defined on the NonProfitOrganization table (derived from of the Organization table) as follows:
    a. Right-click the **Order By** node, and then click **New Field**.
    b. In the property window, select **NonProfitOrganization** for the **TableSelector** property. Select **AnnualContribution** for the **Field** property.

8. Save the query.



```
QueryOrgCustomer(usr)
[+] Methods
[-] Data Sources
    [-] organization_1(organization)
        [ ] Fields
        [-] Ranges
                State
        [-] Data Sources
            [-] Customer_1(Customer)
                [+] Fields
                [+] Ranges
                [+] Data Sources
                [-] Relations
                        organization_1.RecId == Customer_1.Party
        [+] Group By
        [+] Having
        [-] Order By
                organization_1.Name
                (NonProfitOrganization)organization_1.AnnualContribution
[+] Dependent Objects
[+] Composite Query
```

Name	**State**
Table	organization
TableSelector	
Field	**State**
Value	
Label	
Status	Open
Enabled	Yes

Ad hoc query mode

When querying data from the table inheritance hierarchy, Microsoft Dynamics AX 2012 navigates the hierarchy and constructs appropriate SQL queries that join the necessary tables across the hierarchy to retrieve the data. This process can become very expensive if the number of tables in the inheritance hierarchy is large. Developers are encouraged to adopt the *ad hoc query mode* when designing their queries; in other words, they should restrict the field list to only those fields required for the business logic.

When querying the table inheritance hierarchy by using the X++ **select** statement, use **select** *field list* instead of **select *** to improve performance. The following code example demonstrates this difference.

The first X++ select statement in the **SelectCustomer_adhoc** code example specifies including all fields in the returned record. This results in Microsoft Dynamics AX 2012 sending a SQL query, shown after the code example, to the back-end database to join the Party, Organization, NonProfitOrganization, and GovernmentOrganization tables.

```
static void SelectCustomer_adhoc(Args _args)
{
    Organization Org;
    customer     cust;
    boolean          checkFieldSelect=true;

    //Party, Organization, GovernmentOrganization
    // and NonProfitOrganization tables will be joined in the SQL query.
    select * from Org join cust
                        where
                            Cust.party == Org.recid
                         && Org.NumberOfEmployees>100;

    //Only Party and Organization tables will be joined in the SQL query.
    select name from Org join cust
                        where
                            Cust.party == Org.recid
                         && Org.NumberOfEmployees>100;
    info(Org.name);

    if(checkFieldSelect && !Org.isFieldDataRetrieved("dunsnumber"))
    {
        info("DunsNumber not selected ");
    }
    else
    {
        //Run-time error thrown when accessing field Dunsnumber because it is not selected.
        info(Org.Dunsnumber);
    }

}
```

SQL query

```
SELECT t1.name,
       t1.streetaddress,
       t1.postalcode,
       t1.state,
       t1.country,
       t1.city,
       t1.instancerelationtype,
       t1.phone,
       t1.email,
       t1.modifieddatetime,
       t1.recversion,
       t1.relationtype,
       t1.recid,
       t2.numberofemployees,
       t2.dunsnumber,
       t2.recversion,
       t2.relationtype,
       t2.recid,
       t3.governmentorglevel,
       t3.agencydescription,
       t3.recversion,
       t3.relationtype,
       t3.recid,
       t4.taxexempt,
       t4.purpose,
       t4.annualcontribution,
       t4.recversion,
       t4.relationtype,
       t4.recid,
       t5.accountnumber,
       t5.recid
FROM   party t1
       CROSS JOIN organization t2
                  LEFT OUTER JOIN governmentorganization t3
                    ON ( t2.recid = t3.recid )
                  LEFT OUTER JOIN nonprofitorganization t4
                    ON ( t2.recid = t4.recid )
       CROSS JOIN customer t5
WHERE  ( t2.recid = t1.recid )
       AND ( ( t5.dataareaid = @P1 )
             AND ( ( t5.party = t2.recid )
                   AND ( t2.numberofemployees > @P2 ) ) )
```

Limiting the field list to include only *name* will render the SQL query with only the Party and Organization tables being joined and a much shorter select field list, both of which will improve the performance of the data query.

```
SELECT t1.name,
       t1.recid,
       t1.instancerelationtype,
       t2.recid,
       t3.accountnumber,
       t3.recid
FROM   party t1
       CROSS JOIN organization t2
       CROSS JOIN customer t3
WHERE  ( t2.recid = t1.recid )
       AND ( ( t3.dataareaid = @P1 )
             AND ( ( t3.party = t2.recid )
                   AND ( t2.numberofemployees > @P2 ) ) )
```

Note Accessing fields that are not in the **select** field list from an inheritance table buffer will result in a run-time error, such as the following, stating that the field accessed has not been explicitly selected:

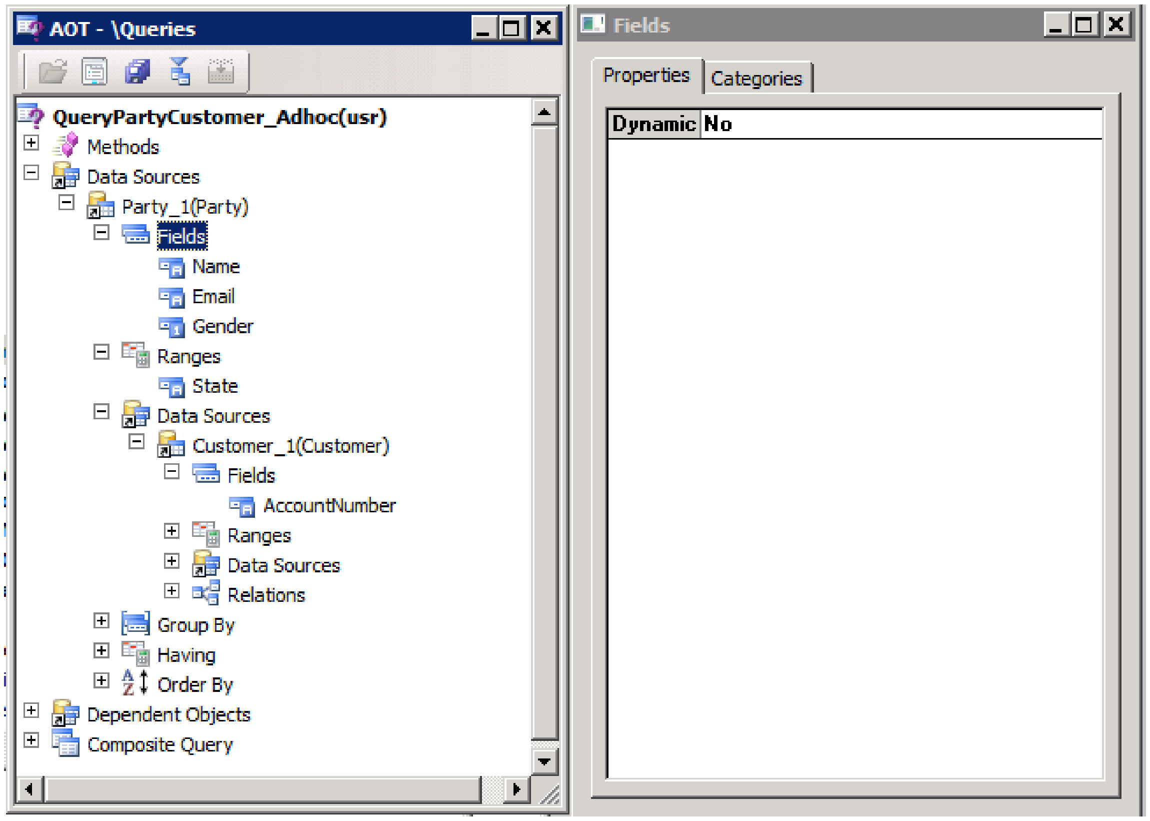
Field 'DunsNumber' in table 'Rel_Organization' has not been explicitly selected.

However, the default behavior when accessing an unselected field from a non-inheritance table buffer is to return the default value of the field data type. To enable the invalid field access check for both inheritance and non-inheritance tables, you need to set the CheckInvalidFieldAccess field in the SysGlobalConfiguration table.

Developers can determine whether a particular field on a table buffer contains data retrieved from the database by calling the **common.IsFieldSelected(str fieldname)** method on the table buffer. This is useful when the table buffer to be accessed is not instantiated within the current function scope.

To enable the ad hoc query mode when designing AOT queries, set the **Data Sources** > **Fields** > **Dynamic** property to **No**, and select only the required fields in the field list (see Figure 8).

Note When a query data source is created (by dragging a table from **AOT** > **Data Dictionary** > **Tables** onto the **Data Sources** node), the default value of the **Dynamic** property is set to **unselect**. The developer must change the default value to either **Yes** or **No**. Otherwise, Microsoft Dynamics AX 2012 will issue a compilation error when the query that has been created is compiled.



```
QueryPartyCustomer_Adhoc(usr)
  Methods
  Data Sources
    Party_1(Party)
      Fields
        Name
        Email
        Gender
      Ranges
        State
      Data Sources
        Customer_1(Customer)
          Fields
            AccountNumber
          Ranges
          Data Sources
          Relations
      Group By
      Having
      Order By
  Dependent Objects
  Composite Query
```

Dynamic	No

Figure 8: Setting the Dynamic property

The *QueryPartyCustomer_adhoc* query demonstrates an AOT query that queries the Person and Organization customers in a particular state and returns the customer's accountnumber, name, email, and gender (if the customer is a person). Note that the developer is able to select the field *gender* from a table Person derived from the data source Party.

The rendered SQL query appears as follows and involves only the required tables in the joins.

```
SELECT t1.email,
       t1.name,
       t1.recid,
       t1.instancerelationtype,
       t2.gender,
       t2.recid,
       t3.accountnumber,
       t3.recid
FROM   party t1
       LEFT OUTER JOIN person t2
         ON ( t1.recid = t2.recid )
       CROSS JOIN customer t3
WHERE  ( t1.state = @P1 )
       AND ( ( t3.dataareaid = @P2 )
             AND ( t3.party = t1.recid ) )
ORDER  BY t1.name
```

Creating a view with the table inheritance hierarchy

Tables that are included in an inheritance hierarchy can be used to create AOT views. The developer should note, however, that Microsoft Dynamics AX 2012 does not navigate to and retrieve data from the hierarchy other than from the table that is selected as the view data source. For example, the view shown in Figure 9 would persist an SQL view without joining the base and derived table of Organization in the hierarchy.

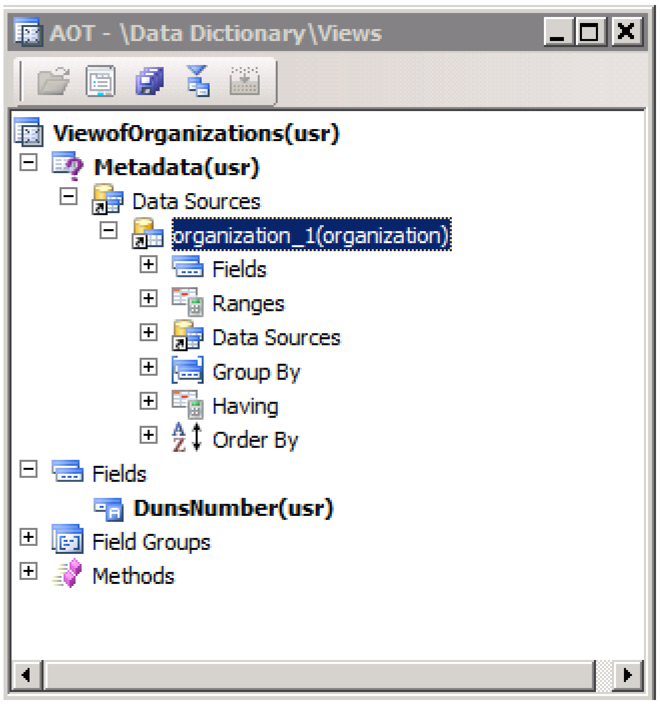


Figure 9: AOT view of Organization table

```
CREATE VIEW "DBO".viewoforganizations
AS
  SELECT t1.dunsnumber AS dunsnumber,
         t1.recid      AS recid
  FROM    organization t1
```

Support for caching

Microsoft Dynamics AX 2012 does not support entire table caching on the table inheritance hierarchy when record-level caching is supported. Record-level caching for a table inheritance hierarchy is enabled by setting the **CacheLookup** table property value on the root table of the hierarchy. Its value governs the caching mode for all tables in the table inheritance hierarchy.

Microsoft Dynamics AX 2012 places a record in the cache when a data query is prescribed a range condition that specifies equal conditions on a set of fields that match the unique key defined on the table. In the case of table inheritance, the unique index and fields in the range condition can come from either the base tables or the derived tables.

System services

In Microsoft Dynamics AX 2012, the services programming model provides support for developing service-based applications that integrate business logic and data with Microsoft Dynamics AX. The table inheritance artifacts are fully exposed and ready to be consumed through the system services. In particular, developers can create and execute queries against the table inheritance hierarchy through the query metadata and query services, and can expect to see the same run-time behavior and programming artifacts that are exposed to X++ developers.

Set-based operations on a table inheritance hierarchy

Microsoft Dynamics AX 2012 does not support using the **RecordInsertList** and **RecordSortedList** classes for bulk inserts into table inheritance hierarchies. Instead, developers can use **insert_recordset** set operation to insert multiple rows of data into a table inheritance hierarchy and save trips to the back-end database.

Tables in the table inheritance hierarchy can serve as either source or destination tables in **insert_recordset** operations. Field inheritance will be honored in the field lists. The developer should refer to http://msdn.microsoft.com/en-us/library/aa635694.aspx for more detail about the use of **insert_recordset**.

Microsoft Dynamics AX 2012 also supports the use of the operations **update_recordset** and **delete_From** to update and delete data stored in the table inheritance hierarchy. However, using **update_recordset** is not allowed for updating a field with an expression that includes fields from another table in the inheritance hierarchy. A compilation error will occur because of this violation.

```
static void AcrossTableUpdate(Args _args)
{
    Rel_Person person;
    //compile error: Field assignment statement cannot involve fields across table
inheritance hierarchy.
    update_recordset person
            setting email = person.email+ person.Prefix
            where person.City = "Seattle";
}
```

Set-based operations will revert to record-by-record operations under certain conditions. These conditions are listed in the following table of supported actions on table inheritance hierarchies.

Note In cases where the target table is in a table inheritance hierarchy, enabling record-by-record fallback conditions on the target table or on any of its base tables will downgrade the set-based operations to record-by-record operations. On the other hand, the **skipxxxx** methods only need to be called on the target table buffer to override any of the fallback conditions set along the inheritance hierarchy.

Condition	delete_From	update_recordset	insert_recordset	Use...to override
Delete actions	Yes	No	No	**skipDeleteActions**
Database log enabled	Yes	Yes	Yes	**skipDatabaseLog**
Overloaded method	Yes	Yes	Yes	**skipDataMethods**
Alerts set up for table	Yes	Yes	Yes	**skipEvents**

Developing forms with table inheritance

With the introduction of table inheritance, developers need to be able to create a form that can display records stored in inheritance tables, with their type varying from record to record. This type of form is called a *polymorphic form*.

Creating a polymorphic form

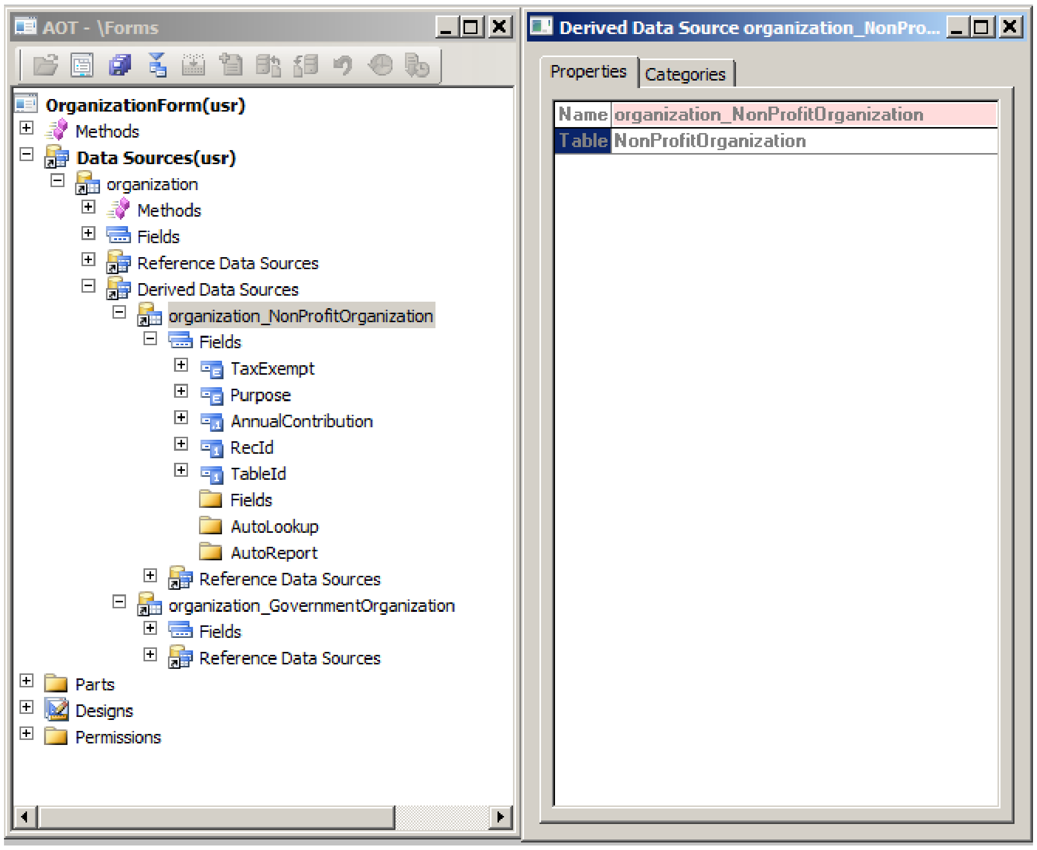
At design time, when an inheritance table is used as a data source to create a polymorphic form, Microsoft Dynamics AX 2012 expands the inheritance hierarchy and creates the derived data sources recursively under the original form data source. Consequently, the fields from the derived data sources can bind to user controls in the form. This feature is demonstrated in the following procedure, which creates a polymorphic form that displays the Organization records from the mock party model (created earlier in this white paper).

To create a polymorphic form

1. Create a new form in the AOT by right clicking **Forms**, and then clicking **New Form**. Name the newly created form **OrganizationForm**.
2. Create a new data source, **Organization,** in the **OrganizationForm** by dragging the Organization table onto the **Forms** > **OrganizationForm** > **Data Sources** node.

   Note Inherited fields from the Party base table are automatically populated under the **Fields** node of the created data source.
3. Expand the **Derived Data Sources** node under the **Organization** data source.

   Note The two data sources under the **Derived Data Sources** node, *organization_NonProfitOrganization* and *organization_GovernmentOrganization,* are automatically created by Microsoft Dynamics AX 2012 for the tables NonProfitOrganization and GovernmentOrganization, which are derived from the Organization table. The **Fields** node under each data source in the **Derived Data Sources** node contains only the fields defined on the derived table.


AOT - \Forms

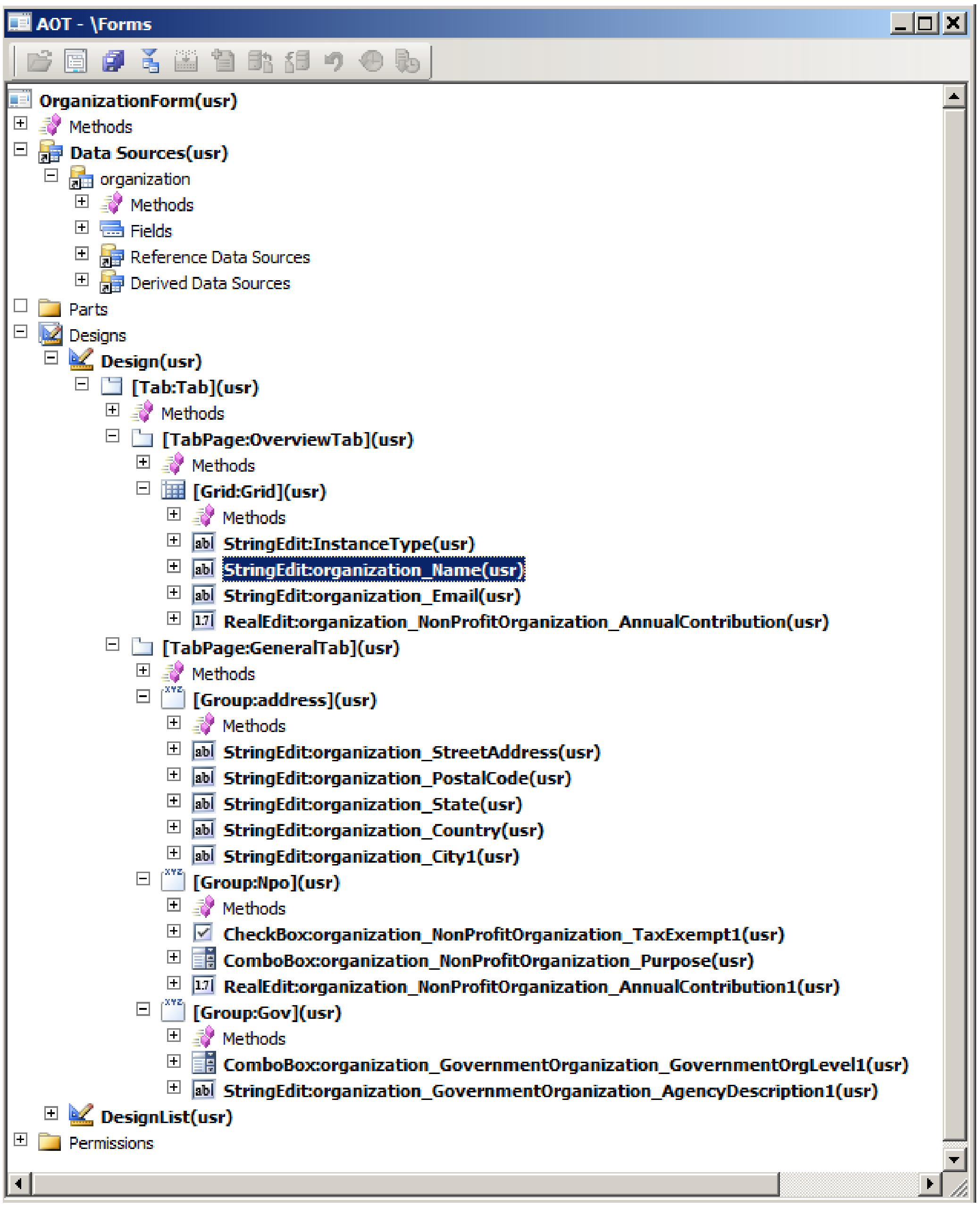
OrganizationForm(usr)
- Methods
- Data Sources(usr)
  - organization
    - Methods
    - Fields
    - Reference Data Sources
    - Derived Data Sources
      - organization_NonProfitOrganization
        - Fields
          - TaxExempt
          - Purpose
          - AnnualContribution
          - RecId
          - TableId
          - Fields
          - AutoLookup
          - AutoReport
        - Reference Data Sources
      - organization_GovernmentOrganization
        - Fields
        - Reference Data Sources
- Parts
- Designs
- Permissions

Derived Data Source organization_NonPro...

Properties | Categories

Name	organization_NonProfitOrganization
Table	NonProfitOrganization

4. Under the **Designs** > **Design** node, create controls binding to the fields from the data source **Organization** and the derived data sources.
5. Set the Form Title data source as follows:
   a. Right-click the **Design** node and then click **Properties**.
   b. In the property window, enter **Organization** as the value of the **TitleDatasource** property. Be sure that the **TitleField1** property is set to **Name** and the **TitleField2** property is set to **City** on the Party table, and that both properties are set to empty on the Organization table.
6. Save the OrganizationForm form.



AOT - \Forms

```
OrganizationForm(usr)
  Methods
  Data Sources(usr)
    organization
      Methods
      Fields
      Reference Data Sources
      Derived Data Sources
  Parts
  Designs
    Design(usr)
      [Tab:Tab](usr)
        Methods
        [TabPage:OverviewTab](usr)
          Methods
          [Grid:Grid](usr)
            Methods
            StringEdit:InstanceType(usr)
            StringEdit:organization_Name(usr)
            StringEdit:organization_Email(usr)
            RealEdit:organization_NonProfitOrganization_AnnualContribution(usr)
        [TabPage:GeneralTab](usr)
          Methods
          [Group:address](usr)
            Methods
            StringEdit:organization_StreetAddress(usr)
            StringEdit:organization_PostalCode(usr)
            StringEdit:organization_State(usr)
            StringEdit:organization_Country(usr)
            StringEdit:organization_City1(usr)
          [Group:Npo](usr)
            Methods
            CheckBox:organization_NonProfitOrganization_TaxExempt1(usr)
            ComboBox:organization_NonProfitOrganization_Purpose(usr)
            RealEdit:organization_NonProfitOrganization_AnnualContribution1(usr)
          [Group:Gov](usr)
            Methods
            ComboBox:organization_GovernmentOrganization_GovernmentOrgLevel1(usr)
            StringEdit:organization_GovernmentOrganization_AgencyDescription1(usr)
    DesignList(usr)
  Permissions
```

Polymorphic form at run time

When the user attempts to create a new record in the polymorphic form, the Microsoft Dynamics AX 2012 runtime prompts the user to select the type of the record to be created by showing a type picker dialog, as shown in Figure 10. The types are the concrete derived table types that derive from the table serving as the data source for the form.

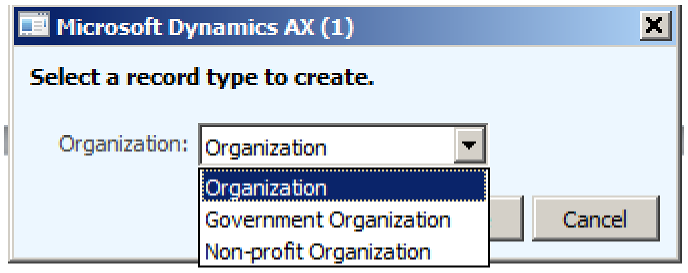
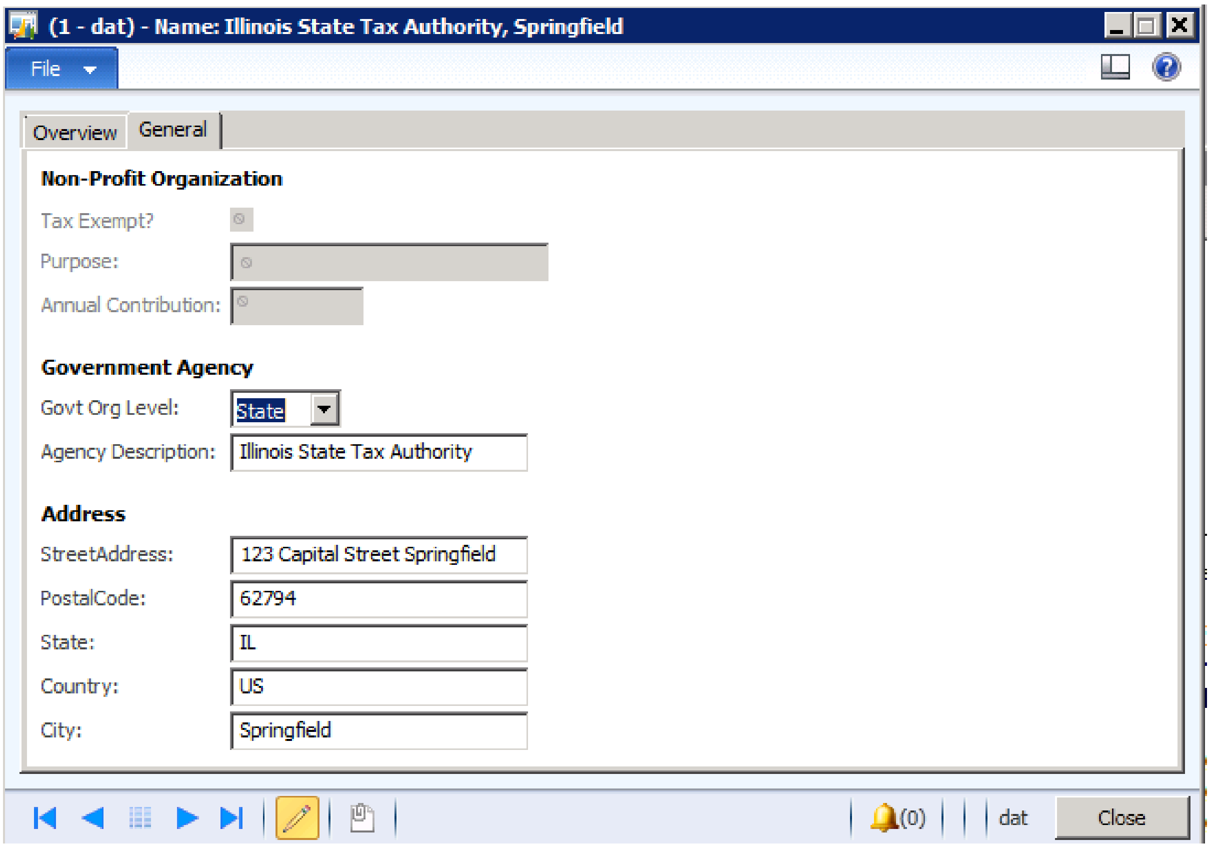


Figure 10: Type picker dialog

Often, you do not want the system default type picker dialog to be displayed when creating new records because of certain constraints imposed by the application requirements. If this is the case, you need to override the **FormRun.CreateRecord** method to replace the default system type picker dialog. In particular, the **FormDatasource.CreateTypes** method can be invoked in the overridden **FormRun.CreateRecord** method to create a new record of the concrete type specified. The developer can reference the SYS layer of the **dirPartyTable** form for the detailed design pattern.

Because the types of records displayed in a polymorphic form vary, Microsoft Dynamics AX 2012 displays visual cues on form controls bound to data fields that are not available. Figure 11 shows that the NonProfitOrganization field-bound edit boxes are disabled when a GovernmentOrganization record “Illinois State Tax Authority” is created.

Note The title fields, **Name** and **City**, displayed in the title bar are defined on the Party table, which is the base type for the Organization data source. The Microsoft Dynamics AX 2012 runtime searches up the inheritance hierarchy, if **TitleField1** and **TitleField2** are not defined on the data source table, to determine the values for these properties.


(1 - dat) - Name: Illinois State Tax Authority, Springfield

Non-Profit Organization

Tax Exempt?	
Purpose:	
Annual Contribution:	

Government Agency

Govt Org Level:	State
Agency Description:	Illinois State Tax Authority

Address

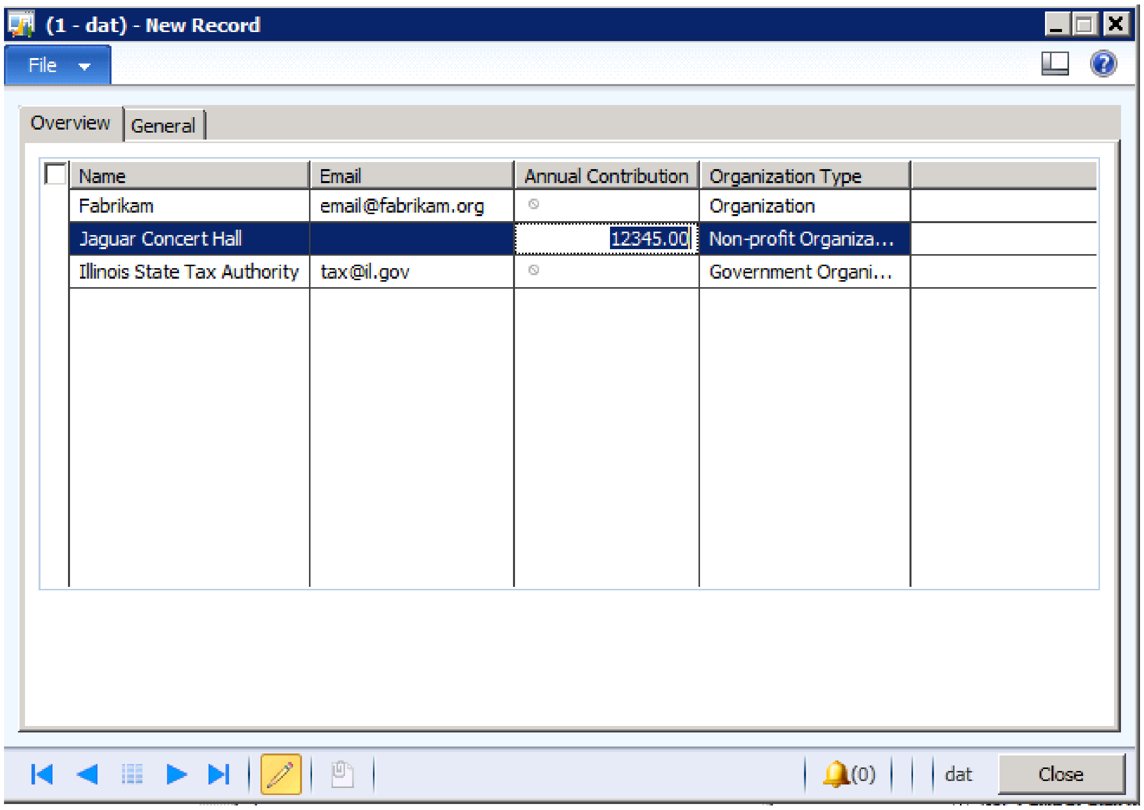
StreetAddress:	123 Capital Street Springfield
PostalCode:	62794
State:	IL
Country:	US
City:	Springfield

Figure 11: Visual cues on form controls bound to data fields that are not available

Displaying the instance type of the record

It is often necessary to display the instance type of the records in the form. We recommend that developers create an enumeration field on the base table to serve as a type discriminator of records. You must override the **insert** data method on the base table to populate the values of this type discriminator when new records are created. This pattern enables localized resources to be displayed when the form is used for a different locale, and allows a form user to sort and filter on the localized type strings.

In the **Organization** form, the **Organization Type** column in the grid is bound to the DisplayRelationType field defined on the Party table and displays the organization type of the associated record (see Figure 12).



Name	Email	Annual Contribution	Organization Type
Fabrikam	email@fabrikam.org		Organization
Jaguar Concert Hall		12345.00	Non-profit Organiza…
Illinois State Tax Authority	tax@il.gov		Government Organi…

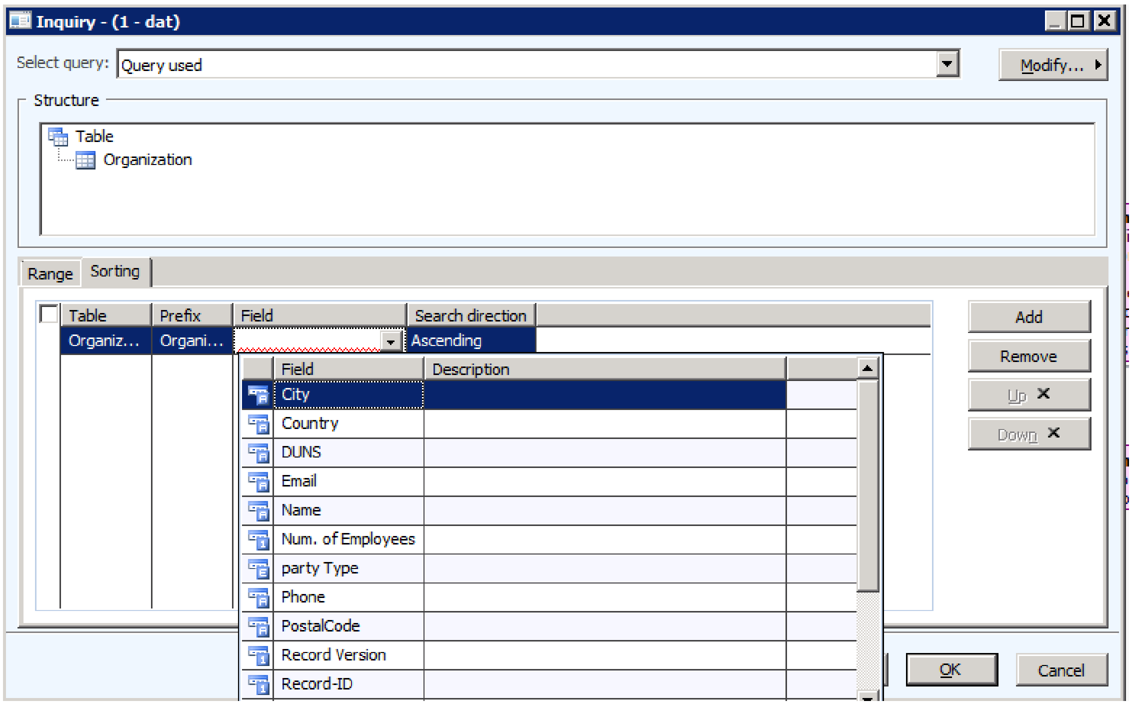
Figure 12: Organization form with Organization Type column bound to DisplayRelationType field on Party table

Record templates on polymorphic forms

Users can create and apply record templates on polymorphic forms. Record templates are created per concrete type and only apply when new records of the concrete type are created.

Advanced filter support on the table inheritance hierarchy

In Microsoft Dynamics AX 2012, the advanced filter is enhanced to work with table inheritance. Users can directly select inherited fields from the base tables to define range filters and sorting conditions (see Figure 13).


Inquiry - (1 - dat)

Select query: Query used

Structure

Table

Organization

Range | Sorting

Table	Prefix	Field	Search direction
Organiz…	Organi…		Ascending

Field	Description
City	
Country	
DUNS	
Email	
Name	
Num. of Employees	
party Type	
Phone	
PostalCode	
Record Version	
Record-ID	

Figure 13: Advanced filtering

In addition, users can define range filters and sorting conditions on fields defined on the derived tables. To do so, the user must first select the derived table from the **Prefix** combo box and then pick the field defined on the derived table (see Figure 14).

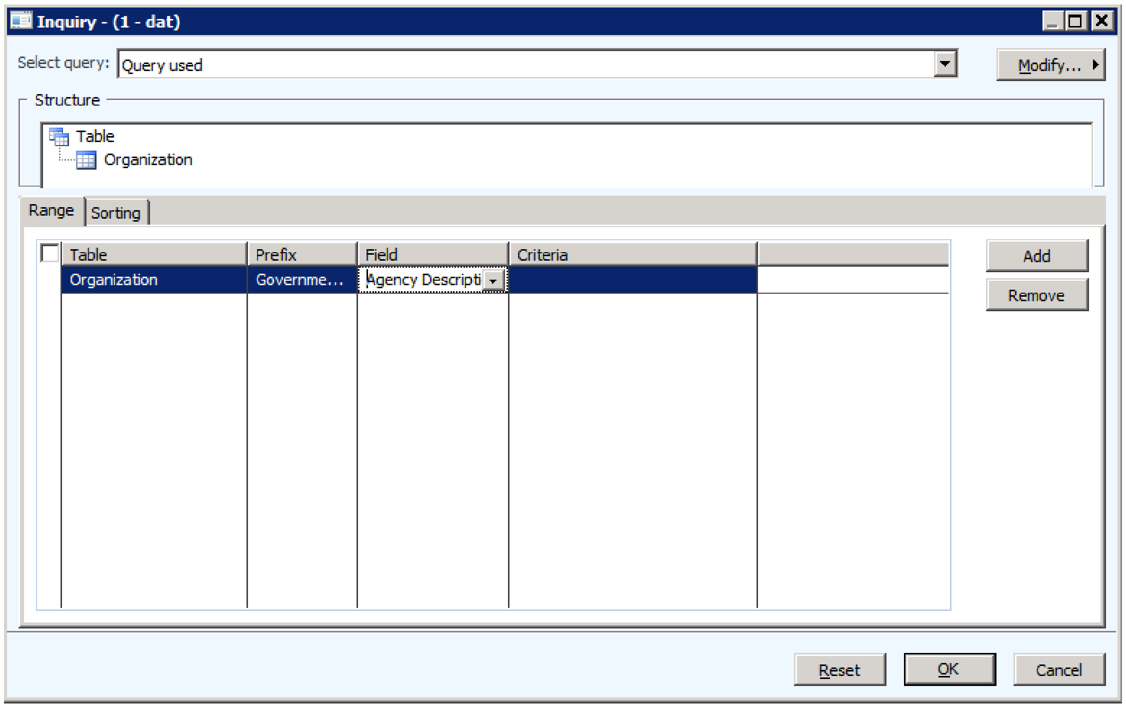


Table	Prefix	Field	Criteria
Organization	Governme...	Agency Descripti	

Figure 14: Advanced filtering (continued)

Using the ad hoc query mode to improve performance

When the ad hoc query mode is applicable and used, the performance of data queries on forms benefits from reduced SQL joins. The ad hoc query mode on the form can be enabled by setting the value of the **OnlyFetchActive** property on the form data source to **Yes**. An **OnlyFetchActive** form data source queries the back-end database for only those fields that are bound to form controls, which improves performance by reducing the SQL joins to the extra tables. These SQL joins can pose a significant burden when the inheritance hierarchy contains large number of tables.

Note When a form is created through a modeled query approach, that is, by dragging an AOT query onto the form to create form data sources, the ad hoc query mode is only enabled when the property **OnlyFetchActive** is set to **Yes** and the query data source field list **Dynamics** property is set to **No**. Both properties must be set.

Developers should use the **OnlyFetchActive** form data source with caution. In general, the ad hoc query mode should only be used on read-only forms. In addition, the business logic code behind the form might access fields that are not bound to controls. Accessing these fields on form data sources that are from inheritance tables will result in the "invalid field" access run-time error. However, accessing an unselected field when the data source is not an inheritance table will only return the default value of the field data type.

To enable checking for invalid field access on all tables, the CheckInvalidFieldAccess field on the SysGlobalConfiguration table needs to be set.

A developer can explicitly add additional non-control–bound fields by overriding the **formdatasource.init** method and inserting the needed fields into the field list as follows.

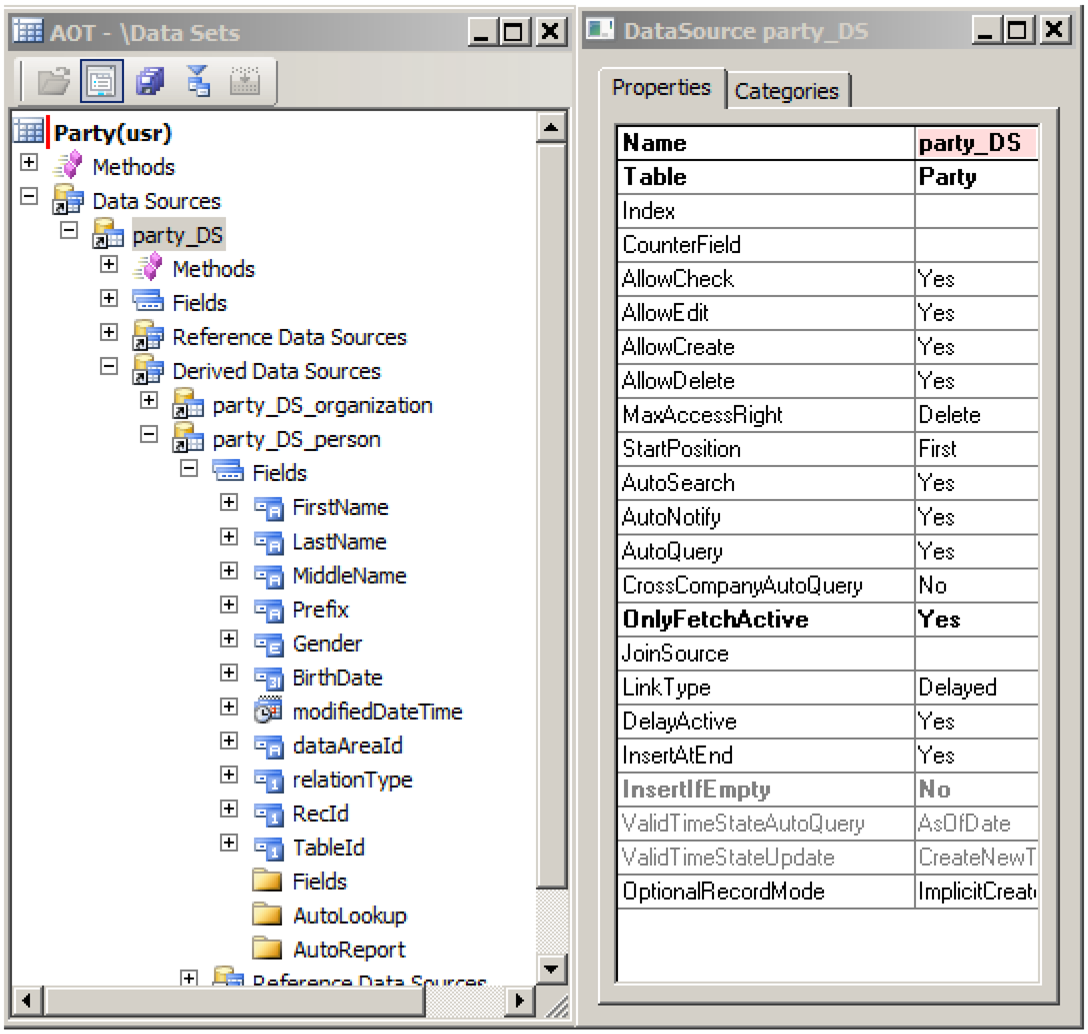
```
public void init()
{
    super();
   this.query().dataSourceTable(tableNum(organization)).fields().addField(
fieldNum(Organization, Dunsnumber));

}
```

Developing Enterprise Portal for Microsoft Dynamics AX 2012 components with table inheritance

In Microsoft Dynamics AX 2012, Enterprise Portal developers can use table inheritance to develop Enterprise Portal components. We demonstrate this capability by creating a web control that displays the entities in the mock party model. Enterprise Portal developers will note that the experience is similar that of form developers.

The data access logic of web controls is defined in the AOT element *Dataset,* which integrates data sources and programming models. When an inheritance table is defined as a data source in a dataset, Microsoft Dynamics AX expands its derived tables recursively and creates derived data sources.

The dataset data source also contains a property, **onlyFetchActive**, which acts the same way as in a form data source to require the Microsoft Dynamics AX 2012 runtime to employ ad hoc query mode when retrieving the data (see Figure 15).

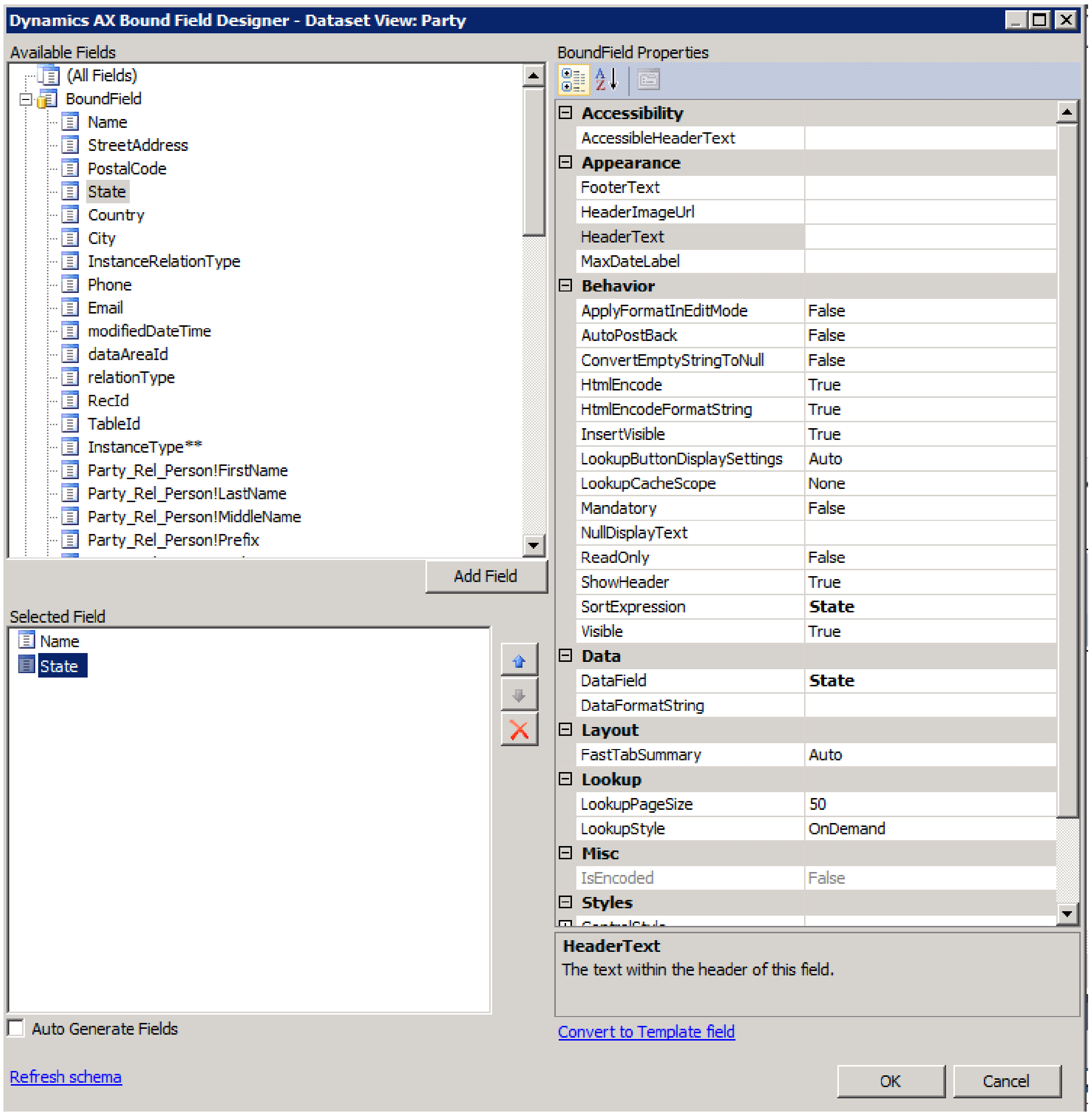


```
Party(usr)
  Methods
  Data Sources
    party_DS
      Methods
      Fields
      Reference Data Sources
      Derived Data Sources
        party_DS_organization
        party_DS_person
          Fields
            FirstName
            LastName
            MiddleName
            Prefix
            Gender
            BirthDate
            modifiedDateTime
            dataAreaId
            relationType
            RecId
            TableId
          Fields
          AutoLookup
          AutoReport
          Reference Data Sources
```

Name	**party_DS**
Table	**Party**
Index	
CounterField	
AllowCheck	Yes
AllowEdit	Yes
AllowCreate	Yes
AllowDelete	Yes
MaxAccessRight	Delete
StartPosition	First
AutoSearch	Yes
AutoNotify	Yes
AutoQuery	Yes
CrossCompanyAutoQuery	No
OnlyFetchActive	**Yes**
JoinSource	
LinkType	Delayed
DelayActive	Yes
InsertAtEnd	Yes
InsertIfEmpty	**No**
ValidTimeStateAutoQuery	AsOfDate
ValidTimeStateUpdate	CreateNewT
OptionalRecordMode	ImplicitCreat

Figure 15: Automatic expansion of the base table data source and derived table data sources

Subsequently, the fields of the derived data sources are listed in the designer when the field bindings on the user controls are defined in Microsoft Visual Studio® (see Figure 16).


Dynamics AX Bound Field Designer - Dataset View: Party

Available Fields

- (All Fields)
- BoundField
  - Name
  - StreetAddress
  - PostalCode
  - State
  - Country
  - City
  - InstanceRelationType
  - Phone
  - Email
  - modifiedDateTime
  - dataAreaId
  - relationType
  - RecId
  - TableId
  - InstanceType**
  - Party_Rel_Person!FirstName
  - Party_Rel_Person!LastName
  - Party_Rel_Person!MiddleName
  - Party_Rel_Person!Prefix

Add Field

Selected Field

- Name
- State

Auto Generate Fields

Refresh schema

BoundField Properties

Accessibility	
AccessibleHeaderText	
Appearance	
FooterText	
HeaderImageUrl	
HeaderText	
MaxDateLabel	
Behavior	
ApplyFormatInEditMode	False
AutoPostBack	False
ConvertEmptyStringToNull	False
HtmlEncode	True
HtmlEncodeFormatString	True
InsertVisible	True
LookupButtonDisplaySettings	Auto
LookupCacheScope	None
Mandatory	False
NullDisplayText	
ReadOnly	False
ShowHeader	True
SortExpression	**State**
Visible	True
Data	
DataField	**State**
DataFormatString	
Layout	
FastTabSummary	Auto
Lookup	
LookupPageSize	50
LookupStyle	OnDemand
Misc	
IsEncoded	False
Styles	

HeaderText
The text within the header of this field.

Convert to Template field

OK Cancel

Figure 16: Fields of the derived data sources listed in the designer

Developing Reporting Services reports with table inheritance

Developers use the Visual Studio integrated development environment (IDE) to create a Microsoft SQL Server Reporting Services report in Microsoft Dynamics AX. These reports can connect to a variety of different types of data sources including Microsoft Dynamics AX and Microsoft Dynamics AX online analytical processing (OLAP). This section demonstrates how to connect to Microsoft Dynamics AX 2012 as the data source and use a query as the data source type to create a report showing the organization entities in the mock party model.

The **QueryOrg** query shown in Figure 17 defines the data access logic on which the Organization report will be based.

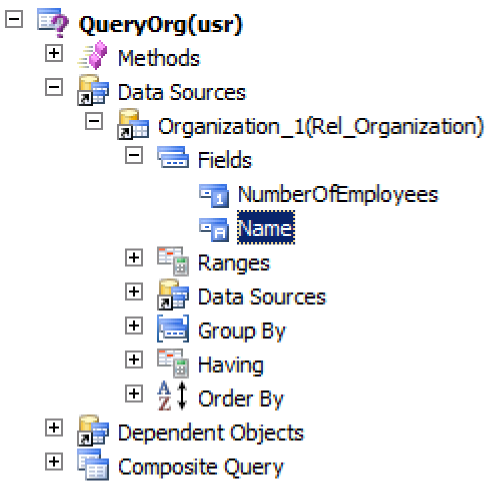
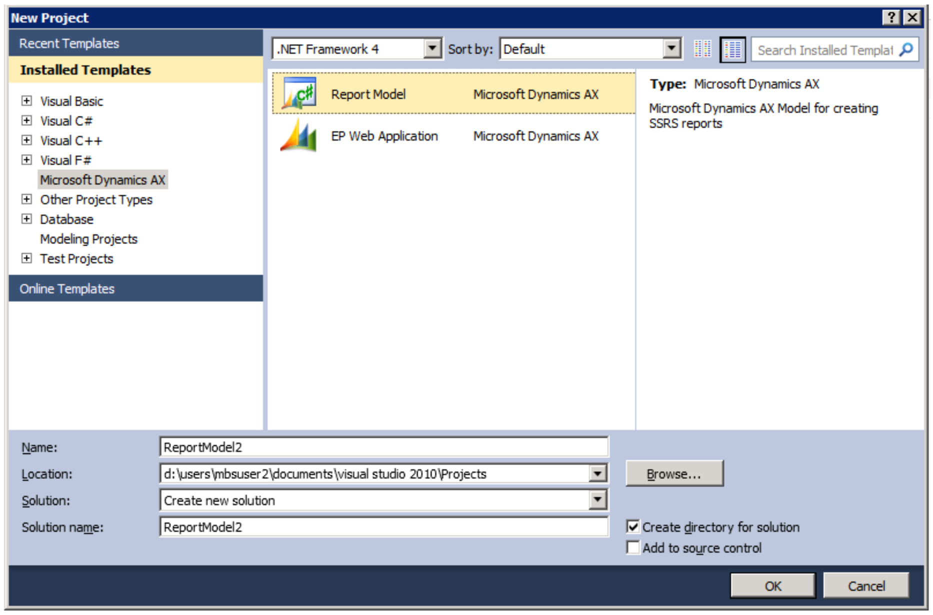


Figure 17: QueryOrg query

To create a report with table inheritance

1. Open Visual Studio and create a new project by using the Microsoft Dynamics AX Report Model template.



Type: Microsoft Dynamics AX

Microsoft Dynamics AX Model for creating SSRS reports

Name: ReportModel2

Location: d:\users\mbsuser2\documents\visual studio 2010\Projects

Solution: Create new solution

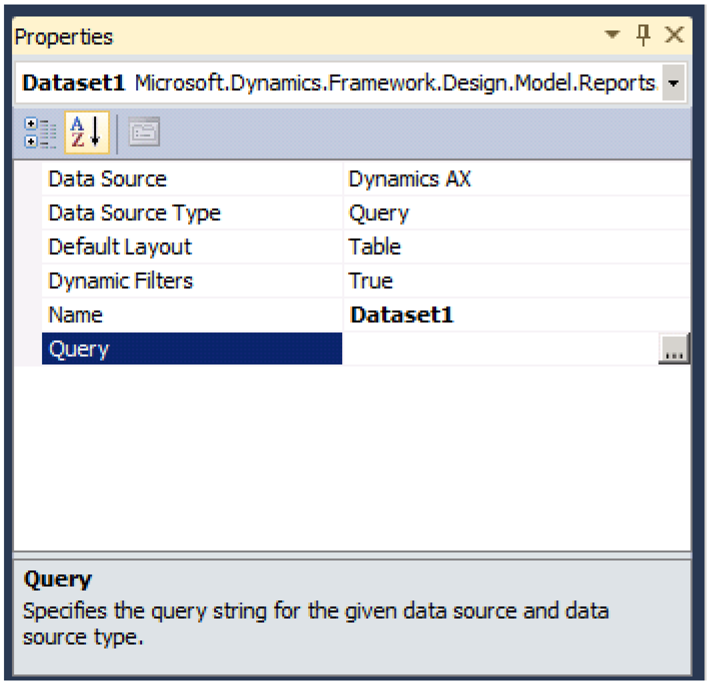
Solution name: ReportModel2

[✓] Create directory for solution

[ ] Add to source control

2. Create a report and add a dataset as follows:

   a. Right click the **Project** node in the Solution Explorer and click **Add > Report**.
   b. Right click the **Datasets** node under the created report and click **Add Dataset**.
   c. In the properties window of the created dataset, choose **Microsoft Dynamics AX** as the data source, and **Query** as the data source type.



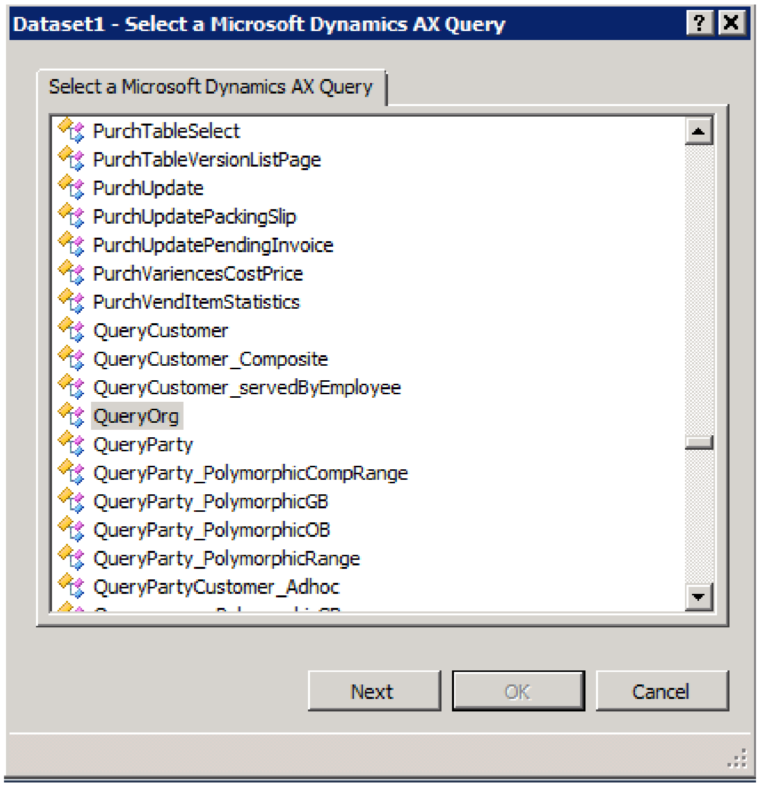
Dataset1 Microsoft.Dynamics.Framework.Design.Model.Reports

Data Source	Dynamics AX
Data Source Type	Query
Default Layout	Table
Dynamic Filters	True
Name	**Dataset1**
Query	

Query
Specifies the query string for the given data source and data source type.

d. Click the button in the **Query** property to launch the **Select a Microsoft Dynamics AX Query** dialog.

e. Select **QueryOrg** from the query list in the **Select a Microsoft Dynamics AX Query** dialog and click **Next**.

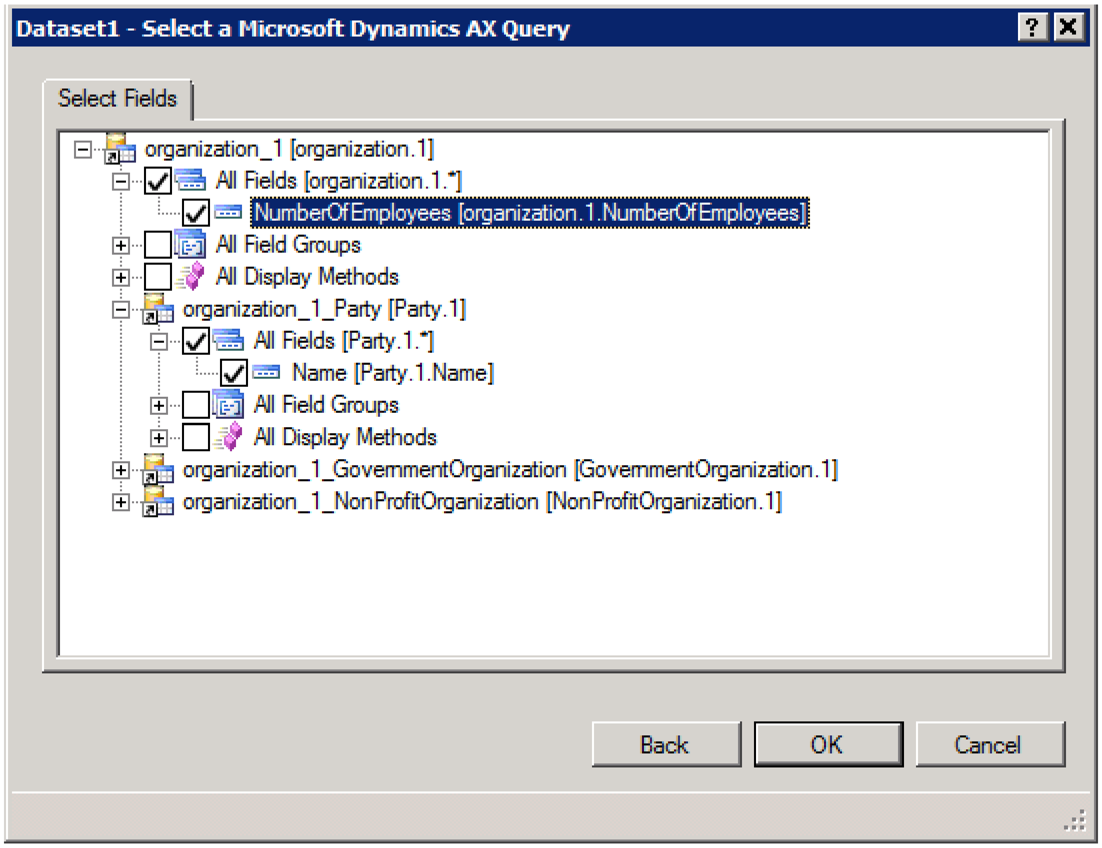


Dataset1 - Select a Microsoft Dynamics AX Query
Select a Microsoft Dynamics AX Query
Next
OK
Cancel

PurchTableSelect
PurchTableVersionListPage
PurchUpdate
PurchUpdatePackingSlip
PurchUpdatePendingInvoice
PurchVariencesCostPrice
PurchVendItemStatistics
QueryCustomer
QueryCustomer_Composite
QueryCustomer_servedByEmployee
QueryOrg
QueryParty
QueryParty_PolymorphicCompRange
QueryParty_PolymorphicGB
QueryParty_PolymorphicOB
QueryParty_PolymorphicRange
QueryPartyCustomer_Adhoc

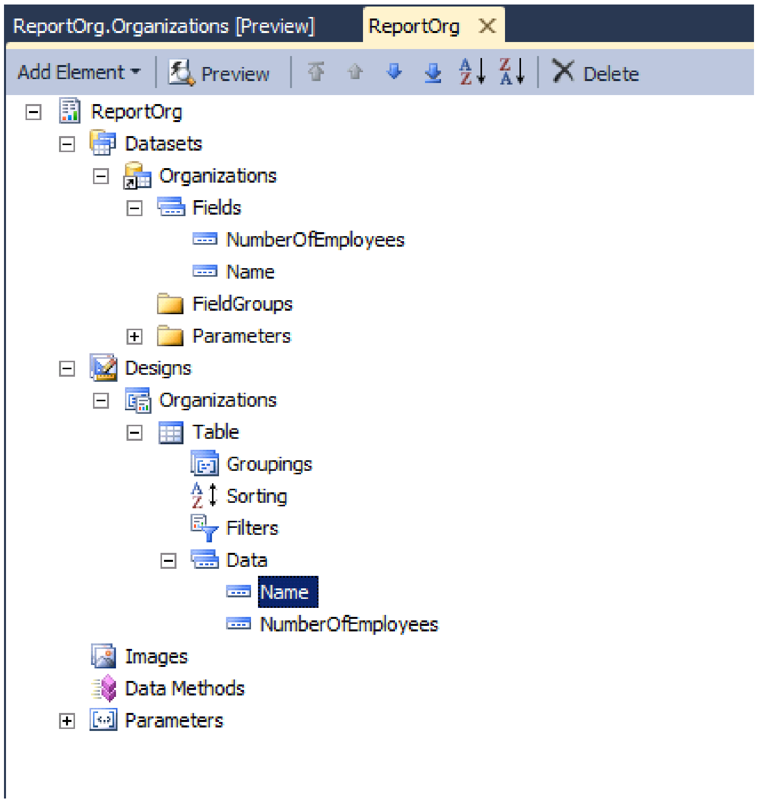
f. Select the fields **Name** and **NumberOfEmployees**.

g. Click **OK**.



- organization_1 [organization.1]
  - ✓ All Fields [organization.1.*]
    - ✓ NumberOfEmployees [organization.1.NumberOfEmployees]
  - All Field Groups
  - All Display Methods
  - organization_1_Party [Party.1]
    - ✓ All Fields [Party.1.*]
      - ✓ Name [Party.1.Name]
    - All Field Groups
    - All Display Methods
  - organization_1_GovernmentOrganization [GovernmentOrganization.1]
  - organization_1_NonProfitOrganization [NonProfitOrganization.1]

3. To add the report to the AOT, create an **Autodesign** by right clicking the **Designs** node, clicking **Add > AutoDesign**, and specifying "Organizations" as the name.
4. Add a **Table** node to the design.
5. Drag the **NumberOfEmployees** and **Name** fields from the **Organizations** node in **DataSets** onto the **Data** node of the table.



The developer of a Reporting Services report does not need to explicitly turn on the ad hoc query mode to have access to the data sources for the report. The Microsoft Dynamics AX 2012 runtime always retrieves data from the table inheritance hierarchy in the ad hoc query mode.

The following SQL query demonstrates the ad hoc mode query sent to the database when the Organizations report is rendered.

```
SELECT t1.name,
       t1.recid,
       t1.instancerelationtype,
       t2.numberofemployees,
       t2.recid
FROM   party t1
       CROSS JOIN organization t2
WHERE (t2.recid = t1.recid)
ORDER  BY t1.name
```

Security and table inheritance

In Microsoft Dynamics AX 2012, the security-key–based security model has been replaced with a role-based security model. It is assumed that the readers of this paper already understand the role-based security model.

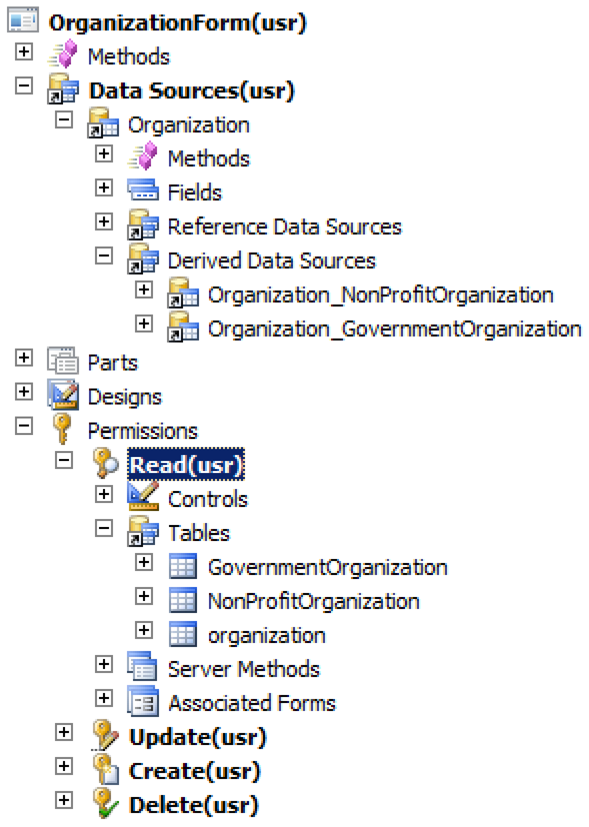
Securing concrete types

It is important to understand that the role-based security model in Microsoft Dynamics AX 2012 grants permissions to only concrete type tables in the table inheritance hierarchy. Permissions that are granted to a concrete type table pertain to the particular type only and do not propagate to its base and derived types.

Values for the **AOSAuthorization** property on the tables in an inheritance hierarchy can be configured differently across the hierarchy and apply to the concrete type represented by the table. The Microsoft Dynamics AX 2012 role-based security model enforces specified data access permission checks on only those concrete table types that have the table permission check enabled (in other words, their **AOSAuthorization** property is set to a value other than "none").

Note When data is accessed by means of a form, the table permission check is always enabled.

Microsoft Dynamics AX 2012 automatically constructs the permission sets needed to access the set of securables when developers create user interface elements (forms, reports, and so on). Developers should note that when the data source is an inheritance table, the automatically created permission sets will contain permissions to access the table and all of its derived tables. This action is demonstrated in Figure 18, where the polymorphic form **OrganizationForm** has a data source created on the base table Organization of the mock party model hierarchy. Expanding the permission sets generated by the AutoInference feature reveals the permissions for accessing the concrete types in the mock party model.



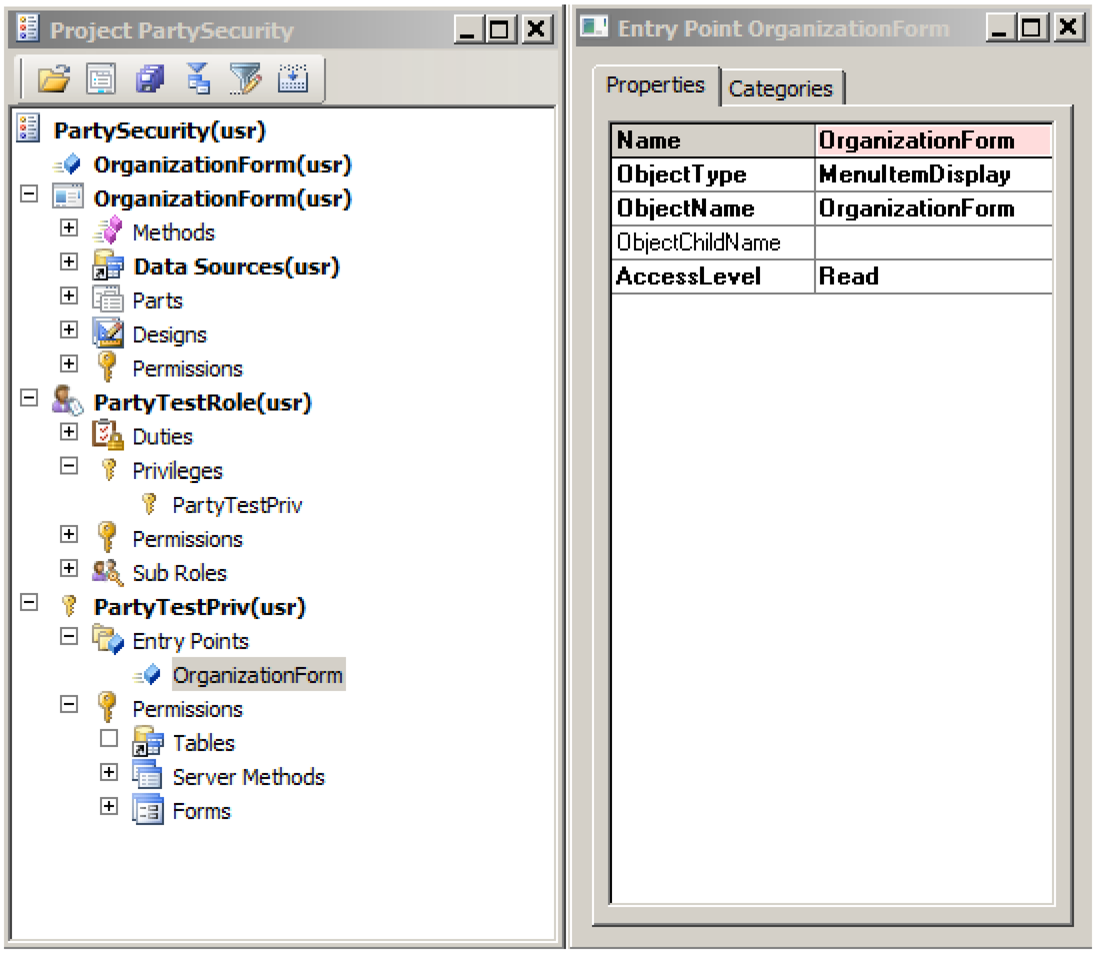
```
OrganizationForm(usr)
  + Methods
  - Data Sources(usr)
      - Organization
          + Methods
          + Fields
          + Reference Data Sources
          - Derived Data Sources
              + Organization_NonProfitOrganization
              + Organization_GovernmentOrganization
  + Parts
  + Designs
  - Permissions
      - Read(usr)
          + Controls
          - Tables
              + GovernmentOrganization
              + NonProfitOrganization
              + organization
          + Server Methods
          + Associated Forms
      + Update(usr)
      + Create(usr)
      + Delete(usr)
```

Figure 18: Automatically created permission sets

Accessing data in the table inheritance hierarchy as a non-administrator

When querying data from an inheritance base table, a user in a non-administrator role will receive only those records from the base type and its derived types that the user has permission to access. This principle can be demonstrated by using the form, **OrganizationForm**, which displays the Organization, NonProfitOrganization and GovernmentOrganization concrete type records stored in the mock party model.

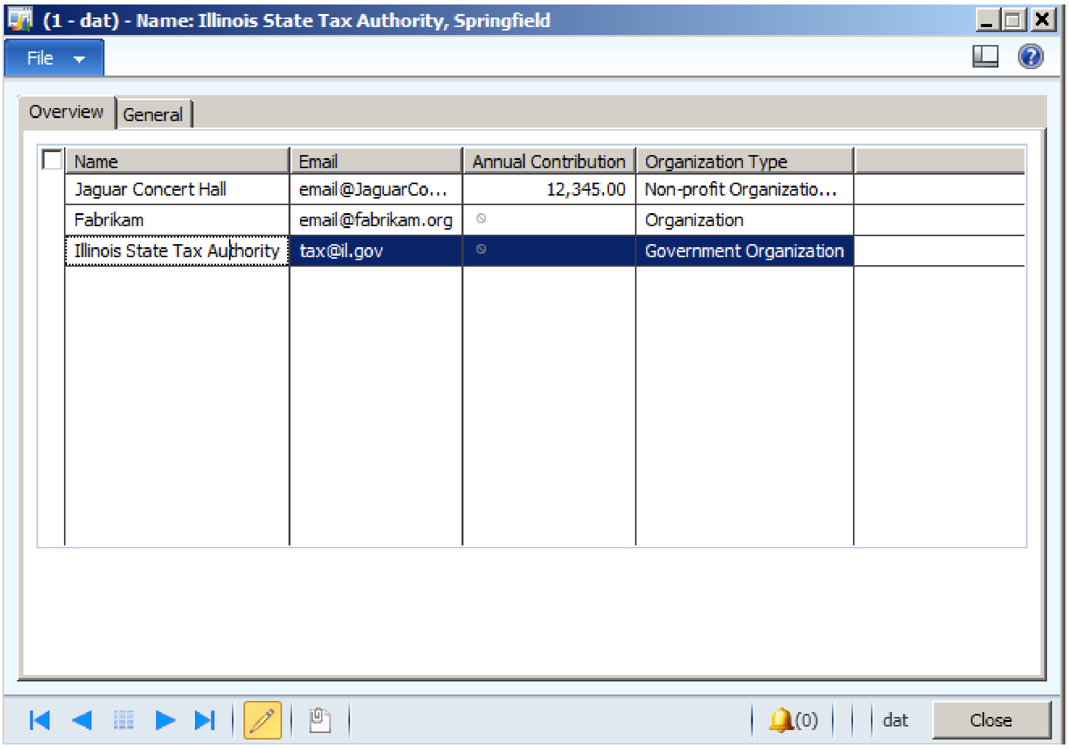
A **PartyTestPriv** privilege grants access to the **OrganizationForm** form, with the read permission set generated by the AutoInference feature. This permits a **read** to be performed on the Organization, NonProfitOrganization and GovernmentOrganization concrete types in the mock party model. A non-administrator is associated with the permission set defined in the **PartyTestPriv1** privilege through the **PartytestRole** role (see Figure 19).



Name	OrganizationForm
ObjectType	MenuItemDisplay
ObjectName	OrganizationForm
ObjectChildName	
AccessLevel	Read

Figure 19: Granting read permissions to a non-administrator

When opened by a non-administrator, the **OrganizationForm** displays read-only records of the concrete type tables: GovernmentOrganization, NonProfitOrganization, and Organization (see Figure 20).



Name	Email	Annual Contribution	Organization Type
Jaguar Concert Hall	email@JaguarCo...	12,345.00	Non-profit Organizatio...
Fabrikam	email@fabrikam.org		Organization
Illinois State Tax Authority	tax@il.gov		Government Organization

Figure 20: OrganizationForm form with read-only records of all concrete type tables

Permission sets generated by the AutoInference feature can be modified, in place, or overridden. You can override them in either the **PartyTestPriv** privilege or the **PartytestRole** role to change the effective permissions on the concrete types.

Suppose an organization decides that a non-administrator cannot have access to GovernmentOrganization records or to the **Email** data column of the NonProfitOrganization records, as shown in Figure 21. Denying access is accomplished by overriding the permissions in the **PartytestRole** role.

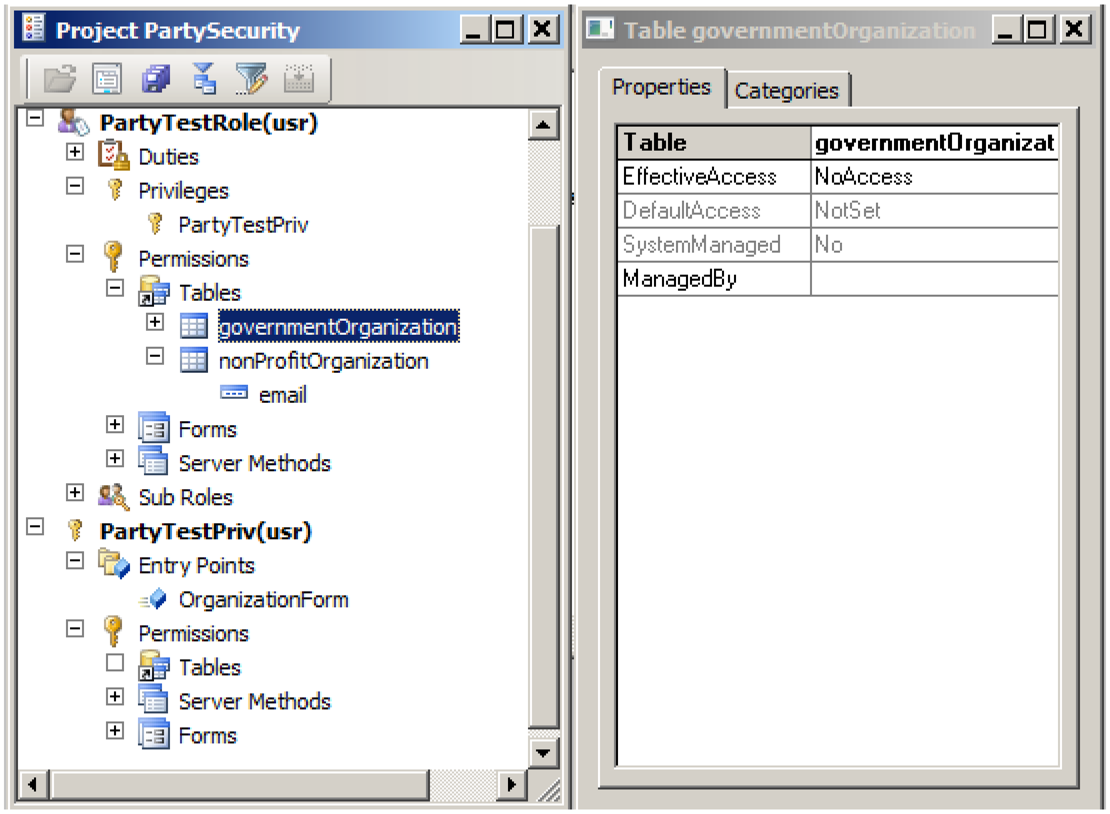
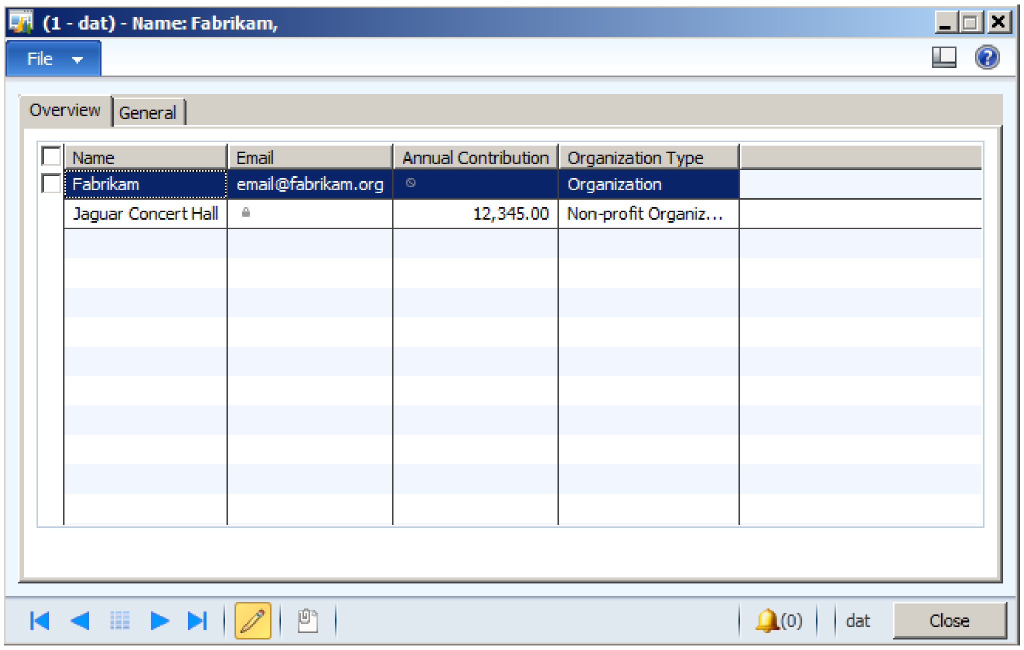


Table	governmentOrganizat
EffectiveAccess	NoAccess
DefaultAccess	NotSet
SystemManaged	No
ManagedBy	

Figure 21: Denying a non-administrator access to GovernmentOrganization records

When opened by a non-administrator with the updated permission set, GovernmentOrganization records will be completely excluded from the records displayed on the **OrganizationForm**. Furthermore, the **Email** data column of the **Non-profit Organization** record type will be protected (see Figure 22).



Name	Email	Annual Contribution	Organization Type
Fabrikam	email@fabrikam.org		Organization
Jaguar Concert Hall		12,345.00	Non-profit Organiz...

Figure 22: Non-administrator with no access to Email column of Non-profit Organization

Data upgrade

Microsoft Dynamics AX 2012 manages data access operations on a live system to ensure that the inheritance relationship is correctly set on the data stored in the hierarchy. However, when performing a data upgrade from a previous version of Microsoft Dynamics AX, the developer is responsible for setting the correct inheritance relationships of the data in the target tables if the data schema of the upgrading solution is being converted to a table inheritance model. These responsibilities include:

- Linking the data stored in the base table and a derived table by setting the **baseTable.RecId=derivedTable.RecId** relationship in the upgrade script.
- Populating the InstanceRelationShip field on the base table with the correct value (the corresponding TableID of the concrete type of the record instance).
- Populating the RelationType fields on each table with the derived TableID that the record spans across. The value is set to **zero** on the table corresponding to the concrete type.

Developers can refer to the upgrade scripts that migrate the global address book data from earlier versions of Microsoft Dynamics AX to Microsoft Dynamics AX 2012 for detailed programming patterns.