Microsoft Dynamics® AX 2012

Implementing and Updating the Human Resources Framework for Microsoft Dynamics AX 2012 Applications

White Paper

This document describes new patterns used to represent workers, legal entities with which workers have employment relationships, and other worker information. It also describes how to convert existing patterns to the Microsoft Dynamics AX 2012 patterns and how to implement the new patterns.

http://microsoft.com/dynamics/ax

Date: April 2011

Author: Andrew Ingalls, Principal Software Architect

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

Microsoft Dynamics

Table of Contents

Overview

In Microsoft Dynamics® AX 2012, the human resources framework has been enhanced to provide greater flexibility and sustainability across the organization. The data model has also been completely revised to permit many of the core Human Resources (HR) tables to be shared and to enable new functionality. Therefore, developers will need to update every reference made to human resources data entities to reference the new tables.

In Microsoft Dynamics AX 2009, multiple worker types were stored in a single table (the EmplTable table), with a type differentiator to indicate whether the entry was for an employee, a contractor, or a work center. Date-effective employment and personal data were stored in a single relationship table, and application teams extended the definition of the EmplTable table by modifying the EmplTable or a related table within the human resources data model. The HRMVirtualNetworkTable table was used to implement functionality common to employees, contractors, applicants, and contacts.

Although the human resources framework in Microsoft Dynamics AX 2012 has been completely redesigned, many of the underlying concepts persist. Microsoft Dynamics AX 2012 still has employees and contractors. Although the "work center" concept does exist, it is no longer a part of the worker data pattern. Additionally, the DirPerson table has been leveraged, eliminating the need for the HRMVirtualNetworkTable table, and data previously associated with that table is now associated with the person, worker, applicant, or contact tables.

You can find a comprehensive list of legacy tables and their replacements in the Appendix. In some cases, there is a one-to-one mapping of the old tables to the new, whereas in other cases—such as the EmplTable table—the data model has undergone extensive normalization. In addition, many of the Human Resources tables that make up workforce management have adopted the kernel-based date-effective framework.

This document does not discuss all of the new functionality within the Human Resources application. Instead, it focuses on development patterns and how they are implemented.

Audience

This document targets developers who are either building new applications for Microsoft Dynamics AX 2012 or updating their existing application code and data.

Terminology

Microsoft Dynamics AX 2012 terms:

Term	Definition
Worker	A person who assumes the role of an employee or a contractor and is paid in exchange for services.
Employee	A role assumed by a person who participates in an employee-employer relationship with a legal entity.
Contractor	A role assumed by a person who participates in a contractor-employer relationship with a legal entity.
Department	A category or functional part of an organization that performs a specific task, such as sales or accounting.

Implementing the human resources framework

This section describes the processes that developers working on new and upgraded code should follow.

New code development

Developers who are writing new code for Microsoft Dynamics AX 2012 that references workers or any of the Human Resources shared components should read the following sections of this white paper:

- Revising your data patterns
- Data model diagrams
- Dialog form controls

Code and data upgrades

Developers who need to perform a code upgrade for existing applications should first identify all of their references to the code patterns described in this white paper, and then follow the instructions in the relevant sections of this white paper to upgrade their code. You can perform a code upgrade in any sequence, but all of the following steps are required:

- Identify the pattern or patterns that your code currently uses.
- Add new fields in the data model to represent the new foreign keys to workers and departments, and to any of the tables that have been replaced (see the Appendix).
- Assign the DEL_ prefix to the old foreign key fields.
- Create a data upgrade script to populate the new fields from the previous fields.
- Update the user interface to use the new control that is appropriate for the pattern defined. The new controls make use of the new foreign keys that you added to your data model.
- Update the references and business logic in your X++ classes, Enterprise Portal pages, and table methods to use the new code patterns.
- Update existing reports to make use of the new data model, including the specific views created for reporting.

Changes to the data model

This section presents a brief description of key tables for Human Resources. It highlights the changes between the previous implementation in Microsoft Dynamics AX 2009, referred to as legacy, and the Microsoft Dynamics AX 2012 implementation. For the physical data model of the tables, see the Appendix.

Worker (HcmWorker) table

Legacy

In Microsoft Dynamics AX 2009, an EmplTable reference, which was typically EmplId, was a string field that held the employee ID and was a foreign key to the EmplTable table. The foreign key relationship was defined as part of the extended data type (EDT). The field was used primarily to identify the employee who was associated with the data being entered. The employee existed within a company and was associated with a position that defined the employee's reporting relationship within the company. The workflow made use of the reporting relationship for the approval hierarchy.

Microsoft Dynamics AX 2012

The worker pattern contains a person, a worker, an employment, and a position assignment. It also normalizes person and worker data into separate tables, based on attribute usage. For example, skills, certificates, and resume data are now associated with the DirPerson table to facilitate movement of people through the hiring process and through any organizational changes that occur during their tenure with the enterprise.

Employment (HcmEmployment) table

Legacy

In Microsoft Dynamics AX 2009, the value of the **SaveDataPerCompany** property of the EmplTable table was always set to **Yes**, which constrained a person's relationship to one company within the enterprise. This limited the ability to reorganize operational organization structures and to support multiple concurrent positions within the organization.

Microsoft Dynamics AX 2012

The worker is a shared resource and can be associated with multiple legal entities. The relationship between a worker and a legal entity is described as being an employment relationship with the legal entity. A worker can have one or more employment relationships with one or more legal entities; each relationship has an independent lifetime. For a data model diagram of the HcmEmployment table, see the Appendix.

Person (DirPerson) table

Legacy

In Microsoft Dynamics AX 2009, the DirPerson table was constrained by DataAreaID (the **SaveDataPerCompany** property was set to **Yes**), but the table could be shared by using the "virtual company" feature.

Microsoft Dynamics AX 2012

A person is a shared entity without a virtual company implementation.

Determining correct table references

Developers will need to determine which association should be held by consumers of the worker pattern (tables that previously used an EmplId field reference in the legacy EmplTable table to identify workers). Although most tables will hold a foreign key reference to the HcmWorker table, a foreign key reference can be held to the following tables, all of which are organizationally independent (that is, the value of their **SaveDataPerCompany** property is set to **No**).

- **HcmWorker –** Contains a foreign key reference to a DirPerson table and contains both employees and contractors.
    - This is the recommended uptake pattern.
    - This table references a worker directly without an association to a specific legal entity.
    - Hold a foreign key to this table when your table needs to reference an employee, a contractor, or both.
    - The table contains two surrogate key field groups, one that allows a person's name to be entered or displayed, and another that enables a personnel number to be entered or displayed.
    - Use the **HcmWorkerRecId** EDT or an existing EDT that extends it. If none of the existing EDTs fits your application needs, a new one can be created by extending **HcmWorkerRecId**.

1. **HcmEmployment –** Contains the relationship between both employees and contractors and a specific legal entity.
    - This uptake pattern should be used when a table holds information that is specific to an employment instance, meaning that the data will expire when employment is terminated.
    - If your table is still constrained by DataAreaId (that is, the **SaveDataPerCompany** property is set to **Yes**), you should also require that the HcmEmployment record be for the same legal entity that is represented by the DataAreaId value in your table record.
    - If you have data that is specific to a legal entity (or DataAreaId), but should exist across employment instances, you should use a foreign key reference to the HcmWorker table and have a **Legal Entity** column or mark the table as **SaveDataPerCompany**=**Yes** (for DataAreaId), and not hold a foreign key to the HcmEmployment table.
2. **DirPerson –** Defines a person.
    - This uptake pattern should be used when the associated data is independent of the role of the person.
    - This pattern does not limit a person to the role of employee. Therefore, all persons will be included in the lookup unless code is added to limit persons to the role of worker. However, to mimic the behavior of the Virtual Network table, there is a specialized lookup that does restrict persons to the role of worker, applicant, or business contact.

In most cases, the EmplID will be replaced with an HcmWorkerRecId field. However, occasionally the replacement will be a DirPersonRecId field or an HcmEmploymentRecId field. For a data model diagram of the HcmWorker subsystem and the Human Resources extensions to the DirPerson table, see the Appendix.

Revising your data patterns

This section highlights new patterns used to represent workers, the legal entities with which workers have established employment relationships, and other worker-related information. It provides details about the new patterns and how to implement them.

Implementing the worker pattern

In Microsoft Dynamics AX 2009, the EmplID from the EmplTable table was used to identify an employee, contractor, or work center in a specific company.

To upgrade your code for Microsoft Dynamics AX 2012, your data model must be replaced with the new field names and extended data types specified in the following table. You need to update the forms where these fields are displayed to use the Worker Lookup control defined in this section. The X++ code that references these fields must use the new fields and code patterns defined for the HcmWorker pattern.

Data model and extended data type

The following table lists the previous and new EDTs and fields related to the worker pattern.

	Previous	Microsoft Dynamics AX 2012
EDTs	EmplId ApprovedBy	HcmWorkerRecId
Fields	EmplId Responsible	Worker Worker is a foreign key to the HcmWorker table.

Updating the code

Replacing the **EmplId** with an **HcmWorkerRecId** requires the following changes throughout the Application Object Tree (AOT):

1. If the needs of your application are not met either by the **HcmWorkerRecId** EDT or by an EDT that extends **HcmWorkerRecId**, create an EDT based on **HcmWorkerRecId** and customize the **Label** and **HelpText** properties.
2. Add a new foreign key relationship to the HcmWorker table on your table, which will add a field and index.

   Note If the EDT or one of its parent EDTs has a table reference defined on it, dropping the EDT onto the table's field list will also create all of the same artifacts (foreign key relationship, field, and index).
3. Validate the relationship properties in the AOT and set them according to the needs of your application.
   a. Determine whether the auto-generated index is needed on your table. If it is not needed, it can be removed.
   b. If your relationship table does not have a mandatory relationship with the HcmWorker table, set the **RelatedTableCardinality** property to **ExactlyOne**, the **Cardinality** property to **ZeroMore**, and the **RelationshipType** property to **Association**. The HcmWorkerBankAccount table is an example of this relationship type.
   c. If your relationship table has a mandatory relationship with the HcmWorker table, set the **RelatedTableCardinality** property to **ExactlyOne**, the **Cardinality** property to **OneMore**, and the **RelationshipType** property to **Association**. You should also perform one of the following actions:
      i. Update the **insert** method on the HcmWorker table and create the mandatory record in your relationship table.
      ii. Update the **createHcmWorker** method in the **HcmWorkerTransition** class and create the mandatory record in your relationship table.
4. Navigate to the new HcmWorker field that was added when the relationship was created and then follow these steps:
   a. Change the value in the **Name** property to the name you have chosen, and set its **ExtendedDataType** property to the extended data type that you chose or created in step 1.
   b. Verify that **Label**, **HelpText**, **Mandatory**, **AllowEditOnCreate**, **AllowEdit** and any other properties set on the replacement field are set correctly by checking their values against those on the old EmplID-based field.

c. If the value of the **Mandatory** property of the old EmplID-based field was set to **Yes**, change the value of the **Mandatory** property to **No.**
d. Determine whether the index that was created when you added the relationship is required. If the index is not required, remove it.

5. If this table existed in Microsoft Dynamics AX 2009, assign the DEL_prefix to the old EmplID-based field rather than removing it from the table.
6. Add the new field to the desired field groups and remove the old EmplID-based field from the group.
7. If the old EmplID-based field in your table is contained in an index for your table, add the new Worker-based field to the index as well.
8. Find all the user interface locations where the EmplID field was used and drop the new Worker-based field (or remove the existing field group and drop the same field group back on the form) to allow the surrogate field substitution to occur.
    - After this action creates the reference group, optionally replace the **AutoIdentification** field group in the **ReplacementFieldGroup** property with the field group you prefer.
9. On the relevant forms, we highly recommend that you implement the **lookupReference** and **resolveReference** methods on the data source for this new field. The next section of this white paper discusses how to implement these methods.
10. Add the DEL_ prefix to old EmplId-based fields and set the **ConfigurationKey** property to **SysDeletedObjects60**.
11. Upgrade your data. See the Creating data upgrade scripts section of this white paper for details.

HcmWorkerLookup form

The **HcmWorkerLookUp** form and the supporting **HcmWorkerLookup** class have been introduced in Microsoft Dynamics AX 2012 to provide a consistent means for establishing a worker relationship while enforcing the business rules of your application. Additional validation helper methods have been added to facilitate data entry without using the lookup.

In Microsoft Dynamics AX 2009, a set of calling forms restricted the EmplTable lookup to display only active records (for employees, contractors, and work centers).

Validation (to permit only active records) of directly entered data was not implemented and could not be specified by using metadata. Therefore, any record in the EmplTable table could be found by typing a value into the field.

By using the **HcmWorkerLookup** class, lookup and validation will use the same logic in Microsoft Dynamics AX 2012.

When to implement the lookupReference and resolveReference methods

Although implementation of the **lookupReference** and **resolveReference** methods is not mandatory, we recommend that you implement them in most cases. If the application restricts data to one or more of the following categories of workers, the **lookupReference** and **resolveReference** methods must be implemented on the form:

- Only workers who are active, terminated, or pending (or a combination of the three)
- Only workers who are employees
- Only workers who are contractors
- Only workers who have active employment in the context of the current legal entity
- Only workers who are Microsoft Dynamics AX users

If you do not override the **lookupReference** and **resolveReference** methods on the form, the default implementation for a field that derives from **HcmWorkerRecId** is to filter the lookup to active workers (both employees and contractors). However, the validate method (**resolveReference**) allows any worker—past, present, or future—to be typed in, which is the same behavior as in Microsoft Dynamics AX 2009.

The following table lists parameters for the **HcmWorkerLookup** implementation class. All parameters take a **Boolean** value. The purpose and impact of the parameters are described in the source code for the class.

Parameters	Default value
includeEmployees	**Yes**
includeContractors	**Yes**
includeActive	**Yes**
includePending	**No**
includeTerminated	**No**
includeOnlyCurrentLegalEntity	**No**
lockWorkerTypeFilters	**No**
lockWorkerStatusFilters	**No**
lockLegalEntityFilter	**No**
requireUserRelation	**No**

These parameters are set on the class instance to enable you to use the same instance for both the **lookupReference** and the **resolveReference** methods. The **HcmWorkerLookup** class also allows the caller to control the visibility of the user interface elements on the lookup form to prevent the end user from changing the filter being applied. If the class instance is used, the settings from the form are retained in the instance. This approach provides the appearance of "remembering" the user's settings. (The settings revert when the master form is closed.) If you use the first code pattern described below for uptake, a class instance is created each time that the method is called. Additionally, the lookup form will always use the defaults for that constructor.

In general, a form is modified to initialize the lookup. There are two ways to implement the changes, as shown in the following examples.

The first option is to initialize the lookup once, and then to reference it from the control on the **Design** node of the form. This optimizes the class instantiation and also allows the **parm** methods to be called on the class to further customize the lookup. This uptake pattern is preferred because it does not reinitialize the class on each lookup or validation. Additionally, it ensures that the same restrictions are used for both the lookup and the validation.

```
public class FormRun extends ObjectRun
{
    Common  callerRecord;

    HcmWorkerLookup     hcmWorkerLookupCurrentCompany;
}

public void init()
{
    super();
```

```
    hcmWorkerLookupCurrentCompany = HcmWorkerLookup::newOnlyActiveEmployeesWithinCompany();
// Could also have called newCustomOptions() to control the lookup/validation behavior.

}
```

On the controls (or fields on the data source), the following methods also need to be added:

```
public Common lookupReference()
{
    Common ret;

    ret = hcmWorkerLookupCurrentCompany.lookupWorker(this);

    return ret;
}

public Common resolveReference()
{
    HcmWorker ret;

    ret = super();

    if (ret != null && !hcmWorkerLookupCurrentCompany.validateWorker(ret.RecId))
    {
        ret = null;
    }

    return ret;
}
```

The second option is to overwrite the methods at the field level on the form data source to invoke the lookup and to validate manual entry. This option is demonstrated in the following example:

```
public Common lookupReference(FormReferenceControl _formReferenceControl)
{
    return
HcmWorkerLookup::newOnlyActiveEmployeesWithinCompany().lookupWorker(_formReferenceControl);
}

public Common resolveReference(FormReferenceControl _formReferenceControl)
{
    Common             ret;

    ret = super(_formReferenceControl);

    if(ret.RecId &&
!HcmWorkerLookup::newOnlyActiveEmployeesWithinCompany().validateWorker(ret.RecId))
    {
        ret.clear();
    }

    return ret;
}
```

Note The HcmWorker and DirPartyTable tables contain many methods that facilitate common tasks in application development. Review the documentation in the source code for method functionality when you develop your own applications.

Implementing the new position pattern

In Microsoft Dynamics AX 2009, the PositionId in the HRPPartyPositionTable table was used to identify an employment position in a company. In Microsoft Dynamics AX 2012, this data is identified by the HcmPosition table.

The HRPPartyPositionTableRelationship table held the details on a position, including which VirtualNetworkReference was currently assigned to it, the position to which this position reported, the department to which the position was assigned, and the job to which this position was assigned.

In Microsoft Dynamics AX 2012, the HRPPartyPositionTableRelationship table has been split into several tables, including the HcmPositionDuration, HcmPositionDetail, HcmPositionDefaultDimension, HcmPositionHierarchy, and HcmPositionWorkerAssignment tables.

Data model and extended data type

In Microsoft Dynamics AX 2009, similar patterns used the same EDT names and field names. When performing uptake, you need to determine which pattern the new fields should use, based on how the data was being used in the old pattern.

	Previous	Microsoft Dynamics AX 2012
EDTs	PositionId	HcmPositionRecId
Fields	PositionId	Position Position is a foreign key to the HcmPosition table.

Updating the code

In Microsoft Dynamics AX 2009, a record in the HRPPartyPositionTableRelationship table had a lifetime (ValidFromDateTime, ValidToDateTime) that effectively defined the lifetime of the attributes for that position. The position itself continued to exist.

In Microsoft Dynamics AX 2012, the HcmPositionDuration table captures the ValidFrom/ValidTo semantics for the position. It is therefore critical to join to the HcmPositionDuration table to validate the existence of the position at a given point in time.

In addition, the attribute tables HcmPositionDetail, HcmPositionHierarchy and HcmPositionWorkerAssignment each have ValidFrom and ValidTo fields that control their respective data. This allows each set of information to be modified independently in its own table. It also allows the developer to choose the information that is important and to retrieve only that data from the table.

HcmPositionLookup form

The **HcmPositionLookUp** form and the **HcmPositionLookup** class have been introduced in Microsoft Dynamics AX 2012 to provide a consistent means of referencing a position and managing the date-effective attributes. Additional validation helper methods have been added to facilitate data entry without use of the lookup.

When to implement the lookupReference and resolveReference methods

Although implementation of the **lookupReference** and **resolveReference** methods is not mandatory, we recommend that you implement them in most cases. If the application restricts data to

one or more of the following position categories, the **lookupReference** and **resolveReference** methods must be implemented on the form:

- Only active positions
- Only filled positions
- Only open positions
- Positions held by a specific worker
- Positions associated with a specific job
- Positions associated with a specific department

If you do not override the **lookupReference** and **resolveReference** methods on the form, the default implementation for a field that derives from **HcmPositionRecId** is to filter the lookup to active positions. However, the validate method (**resolveReference**) allows any position to be typed in.

Parameters passed to the **HcmPositionLookup** implementation class are specified either by the constructor or in the lookup and validate methods.

The constructors allow restrictions on the following parameters, which take **Boolean** values.

Parameter	Default value
includeInactive	**No**
includeOpen	**Yes**
includeFilled	**Yes**

One or both of the *includeOpen* and *includeFilled* parameters must be set to **Yes** (true). It is not valid to ask for positions that are neither open nor filled because there are zero positions that fit that description.

The lookup and validate methods on the class offer additional restriction options as follows:

- Restrict to a specific worker (optional *HcmWorkerRecId*; set to "0" to not restrict)
- Restrict to a specific job (optional *HcmJobRecId*; set to "0" to not restrict)
- Restrict to a specific department (optional *OMOperatingUnitRefRecId*; set to "0" to not restrict)

These parameters are set on the class instance to allow the same instance to be used for both the **lookupReference** and the **resolveReference** methods. If the class instance is used, the settings from the form are retained in the instance. This approach provides the appearance of "remembering" the user's settings. (The settings revert when the master form is closed.) If you use the first code pattern described below for uptake, there is a class instance created each time that the method is called. Additionally, the lookup form will always use the defaults for that constructor.

In general, a form is modified to initialize the lookup. There are several ways to implement the changes, as shown in the following examples.

The first and preferred option is to in initialize the lookup once, and then reference it from the control on the **Design** node of the form. This optimizes the class instantiation, and also allows the **parm** methods to be called on the class to further customize the lookup. This is the preferred uptake pattern because it does not reinitialize the class on each lookup or validation. Additionally, this pattern ensures that the same restrictions are used for both the lookup and the validation.

```
public class FormRun extends ObjectRun
{
    Common  callerRecord;
```

```
    HcmPositionLookup     hcmPositionLookupActive;
}

public void init()
{
    super();

    hcmPositionLookupActive = HcmPositionLookup::newActivePositions(); // Could also have
called newCustomOptions() to control the lookup/validation behavior

}
```

The following methods also need to be added on the controls (or on fields on the data source):

```
public Common lookupReference()
{
    Common ret;

    ret = hcmPositionLookupActive.lookupPosition(this);

    return ret;
}

public Common resolveReference()
{
    HcmWorker ret;

    ret = super();

    if (ret != null && ! hcmPositionLookupActive.validatePosition(ret.RecId))
    {
        ret = null;
    }

    return ret;
}
```

A second option is to override methods at the field level on the data source for the form to invoke the lookup and validate manual entry.

```
public Common lookupReference(FormReferenceControl _formReferenceControl)
{
    return HcmPositionLookup::newActivePositions().lookupPosition(_formReferenceControl);
}

public Common resolveReference(FormReferenceControl _formReferenceControl)
{
    Common             ret;

    ret = super(_formReferenceControl);

    if(ret.RecId && !HcmPositionLookup::newActivePositions().validateWorker(ret.RecId))
    {
        ret.clear();
```

```
        }

        return ret;
    }
```

HcmPositionHierarchy

Several position-related hierarchies can be created. However, there can be only one organizational hierarchy of type **HcmPositionHierarchySystemType::Line**. This particular type is used to determine the management relationships for workers in assigned positions.

The form used to create position hierarchies is **HcmPositionHierarchyView**. This form uses the same managed control as the organization hierarchy form (**HierarchyDesigner**) and therefore uses the **HcmPositionHierarchyViewController** class.

Because of the complexities of multiple position assignments and the date-effective nature of the data, several helper methods for dealing with organizational "reports to" concepts have been created. The methods **getReportsToPosition** and **getReportsToWorker** are defined on the HcmPosition table, and the **getPrimaryPosition** method is defined on the HcmWorker table.

```
public static HcmPositionRecId getReportsToPosition(
        HcmPositionRecId _positionRecId,
        utcdatetime _asOfDate = DateTimeUtil::utcNow(),
        HcmPositionHierarchyTypeRecId _hierarchyTypeRecId =
HcmPositionHierarchyType::lineHierarchyType())

public static HcmWorkerRecId getReportsToWorker(
        HcmPositionRecId _positionRecId,
        utcdatetime _asOfDate = DateTimeUtil::utcNow(),
        HcmPositionHierarchyTypeRecId _hierarchyTypeRecId =
HcmPositionHierarchyType::lineHierarchyType())

public static HcmPositionRecId getPrimaryPosition(
        HcmWorkerRecId _workerRecId,
        utcdatetime _asOfDate = DateTimeUtil::utcNow())
```

Note The HcmWorker and DirPartyTable tables and the **HcmPositionTransition** class contain many methods that facilitate common tasks in application development. Review the documentation in the source code for method functionality when you develop your own applications.

Implementing the shared table natural key replacement

Many tables have been converted in Microsoft Dynamics AX 2012 to allow them to be shared. In this example, the Microsoft Dynamics AX 2009 table HRMCertificateType (now DEL_HRMCertificateType) has been replaced with the new table, HcmCertificateType. For a complete list of legacy tables and their replacements, see the Appendix.

Data model and extended data type

The following table lists the previous and new EDTs and fields related to the shared table natural key replacement pattern.

	Previous	Microsoft Dynamics AX 2012
EDTs	HRMCertificateTypeId	HcmCertificateTypeRecId
Fields	CertificateTypeId	CertificateType CertificateType is a foreign key to the HcmCertificateType table.

Implementation

In simple scenarios, the changes needed on a Microsoft Dynamics AX form are as follows:

1. Modify the referencing table to have a new relationship to the HcmCertificateType table as described in the Updating the code subsection under the Implementing the worker pattern section.
2. Verify that the table that holds the foreign key to the HcmCertificateType table is a data source on the form.
3. Drag the CertificateType field from the data source to the desired location on the form design. This creates a **ReferenceGroup** control with the appropriate **DataSource** and **ReferenceField** property values. Alternatively, you can do this by adding a control to the design and manually setting the **DataSource** and **ReferenceField** properties.

Dialog form controls

The **HcmWorkerUtility** class supports the controls and provides two methods for populating combo boxes on forms with selections that are relevant to a specific worker.

The **getLegalEntitydropdown** method is used to populate a combo box with all legal entities associated with a given worker because of their employment. A detailed description of the method and its parameters are available in code comments in the Microsoft Dynamics AX 2012 source code for this method.

```
public static container getLegalEntitydropdown(
    HcmWorkerRecId          _hcmWorkerRecId,
    FormComboBoxControl     _employmentControl,
    HcmWorkerRelationType   _workerRelationType = HcmWorkerRelationType::Both,
    utcdatetime             _validFrom = DateTimeUtil::utcNow(),
    utcdatetime             _validTo = _validFrom)
```

The following example of how to use this method is taken from the **HcmEmploymentStockOption** form, in which stock options are defined as being specific to a legal entity, and data entry is relevant to the current date.

```
container                   legalEntityRecIdContainer;

legalEntityRecIdContainer = HcmWorkerUtility::getLegalEntitydropdown(
                                                hcmWorker.RecId,
                                                Employment_SelectionLegalEntity,
                                                HcmWorkerRelationType::Employee);
```

A similar method, **populateEmploymentDropDown**, is used for populating a **FormComboBoxControl** with a list of the names of the legal entities to which the user who is currently logged on has access.

```
public static client void populateEmploymentDropDown
(
    FormComboBoxControl     _employmentDropDown,
    container               _companyNameContainer
)
```

Creating data upgrade scripts

A data upgrade script must be created for each legacy table that has been modified. The typical change to a table is to assign the DEL_ prefix to one or more fields (each of which held a foreign key to a now obsolete table), and to add replacement fields that will hold the foreign keys to the new tables in the Microsoft Dynamics AX 2012 data model. The upgrade process will fill in the value of the new field based on the value of the old field.

Note Many examples of upgrade scripts can be found in the class **ReleaseUpdateDB60_HRM**.

The following upgrade code focuses on the worker pattern, where data previously associated with the EmplId field of the EmplTable must now be associated with the HcmWorker table.

Code patterns

The following patterns can be used in the upgrade script:

- **Code pattern using an inner join**

  This code pattern updates the hrmSampleTable table, which previously used the EmplId field (now DEL_EmplId), but which must now hold a field that is a foreign key to the HcmWorker table. It does so by using a join to the EmplTable table and another join to the HcmWorker table. This pattern also performs the update one record at a time by means of the **while select** construct.

```
    HRMSampleTable hrmSampleTable;
    DEL_EmplTable emplTable;
    HcmWorker hcmWorker;

    while select forupdate * from hrmSampleTable
        join Party from emplTable
                where emplTable.EmplId == hrmSampleTable.del_EmplId
        join RecId from hcmWorker
                where hcmWorker.Person == emplTable.Party
    {
        this.tableProgress(tableNum(hrmSampleTable));
        hrmSampleTable.Worker = hcmWorker.RecId;
        hrmSampleTable.update();
}
```

- **Code pattern with a set-based update**

  This pattern accomplishes the same task as the previous pattern. However, it is written to only execute one statement on the SQL database, so it takes less time to execute. The **update_recordset** construct has some limitations that the **while select** construct does not have, so the developer must make an appropriate choice between the two patterns.

```
    hrmSampleTable hrmSampleTable;
    HcmWorker           hcmWorker;
    DEL_EmplTable           emplTable;
```

```
hrmSampleTable.skipDataMethods(true);
hrmSampleTable.skipDatabaseLog(true);

 // Update new FK reference from the HcmWorker table.
 update_recordset hrmSampleTable
     setting Worker = hcmWorker.RecId
 join Party from emplTable
         where emplTable.EmplId == hrmSampleTable.del_hrmEmplId
 join RecId from hcmWorker
     where hcmWorker.Person == emplTable.Party;
```

- **Code pattern with a "while select" in place replacement**

  This pattern may be appropriate when the "inner join" pattern cannot be used. The **while select** statement is only on the hrmSampleTable table and does not include a join to the HcmWorker or DirPerson table. The translation from DEL_EmplId to the RecId column in the HcmWorker table is done in the same line of code, *in place*, as the assignment to the new Worker field.

```
 HRMSampleTable hrmSampleTable;
 DEL_EmplTable           emplTable;

while select * from hrmSampleTable
{
    hrmSampleTable.Worker =
HcmWorker::findByPerson(DEL_EmplTable::find(hrmSampleTable.del_emplId).Party).RecId;
b…other updates;
   }
```

In Microsoft Dynamics AX 2009, if the DirPartyTable table was part of a virtual company, multiple EmplTable records could share the same value in their Party fields.

In Microsoft Dynamics AX 2012, the upgrade creates just one HcmWorker table record in those cases where multiple EmplTable records shared the same value in their Party fields. This leads to some EmplTable records being discarded. The discarded EmplIds are saved in the DEL_EmplTableDiscard table. Some data previously associated with those discarded EmplIds will be upgraded by being associated with the one HcmWorker record.

However, data cannot always be preserved. If the data model for a specific table has changed and now allows only one set of information to be associated with the HcmWorker table, there will be data loss associated with those discarded EmplIds. For example, in Microsoft Dynamics AX 2012, the HcmWorkerTaxInfo table is associated with the HcmWorker table, and only one record is allowed for the worker. In Microsoft Dynamics AX 2009, there could be multiple tax information records for different employees when they shared the same value in their Party fields, due to the virtual company implementation. In this situation, data loss will occur during an upgrade.

- **Code pattern with a "notexists join"**

  This code pattern can be used for the special case just described.

```
while select * from hrmEmplTaxInfo
        join RecId, Party from emplTable
                where emplTable.EmplId == hrmEmplTaxInfo.EmplId
        join RecId from hcmWorker
                where hcmWorker.Person == emplTable.Party
        notexists join emplTableDiscard where emplTableDiscard.DiscardedEmpl ==
emplTable.RecId
{
}
```

Appendix

Shared tables

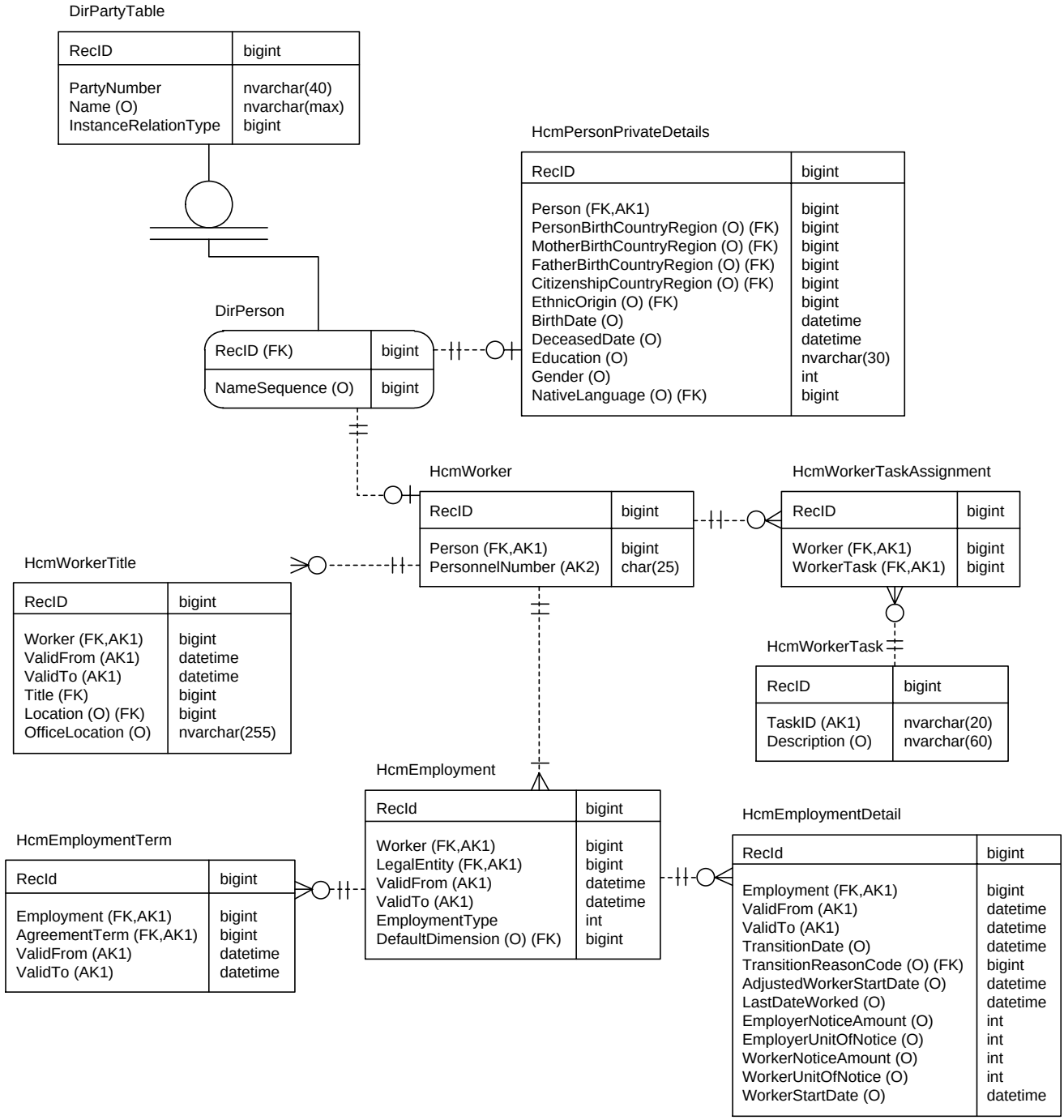
One of the key architectural changes for the new functionality of the Human Resources application in Microsoft Dynamics AX 2012 is the removal of the barrier to data sharing. This restriction was imposed by constraining data to a specific legal entity through the DataAreaID of a table. Many tables, particularly those that contain setup information, have been replaced with new tables that remove this restriction. Developers need to update applications that are associated with any of the Microsoft Dynamics AX 2009 tables listed in the following table to reference the equivalent Microsoft Dynamics AX 2012 table.

Microsoft Dynamics AX 2009 table	Microsoft Dynamics AX 2012 Table
EmplWorkTask	HcmWorkerTask
HRCCompLevel	HcmCompensationLevel
HRMAccomodationType	HcmAccomodationType
HRMBenefitType	HcmBenefitType
HRMCertificateType	HcmCertificateType
HrmCourseType	HcmCourseType
HrmCourseTypeCertificateProfile	HcmCourseTypeCertificateProvile
HrmCourseTypeEducationProfile	HcmCourseTypeEducationProfile
HrmCourseTypeGroup	HcmCourseTypeGroup
HrmCourseTypeSkillProfile	HcmCourseTypeSkillProvile
HRMEducationCenter	HcmEducationInstitution
HRMEducationDegree	HcmEducationLevel
HRMEducationGroup	HcmEducationDisciplineCategory
HRMEducationGroupType	HcmEducationDisciplineGroup
HRMEducationType	HcmEducationDiscipline
HRMEmplCategory	HcmPayrollFrameCategory
HRMEthnicOrigin	HcmEthnicOrigin
HRMGoalHeading	HcmGoalHeading
HRMGoalType	HcmGoalType
HRMGoalTypeTemplate	HcmGoalTypeTemplate
HRMHiringTerms	HcmAgreementTerm
HRMi9DocumentType	Hcmi9DocumentType
HRMi9IssuingAuthority	HcmIssuingAgency
HRMIdentificationType	HcmIdentificationType
HRMIncomeTaxCategory	HcmIncomeTaxCategory
HRMIncomeTaxCode	HcmIncomeTaxCode
HRMInsuranceType	HcmInsuranceType
HRMInterviewType	HcmDiscussionType

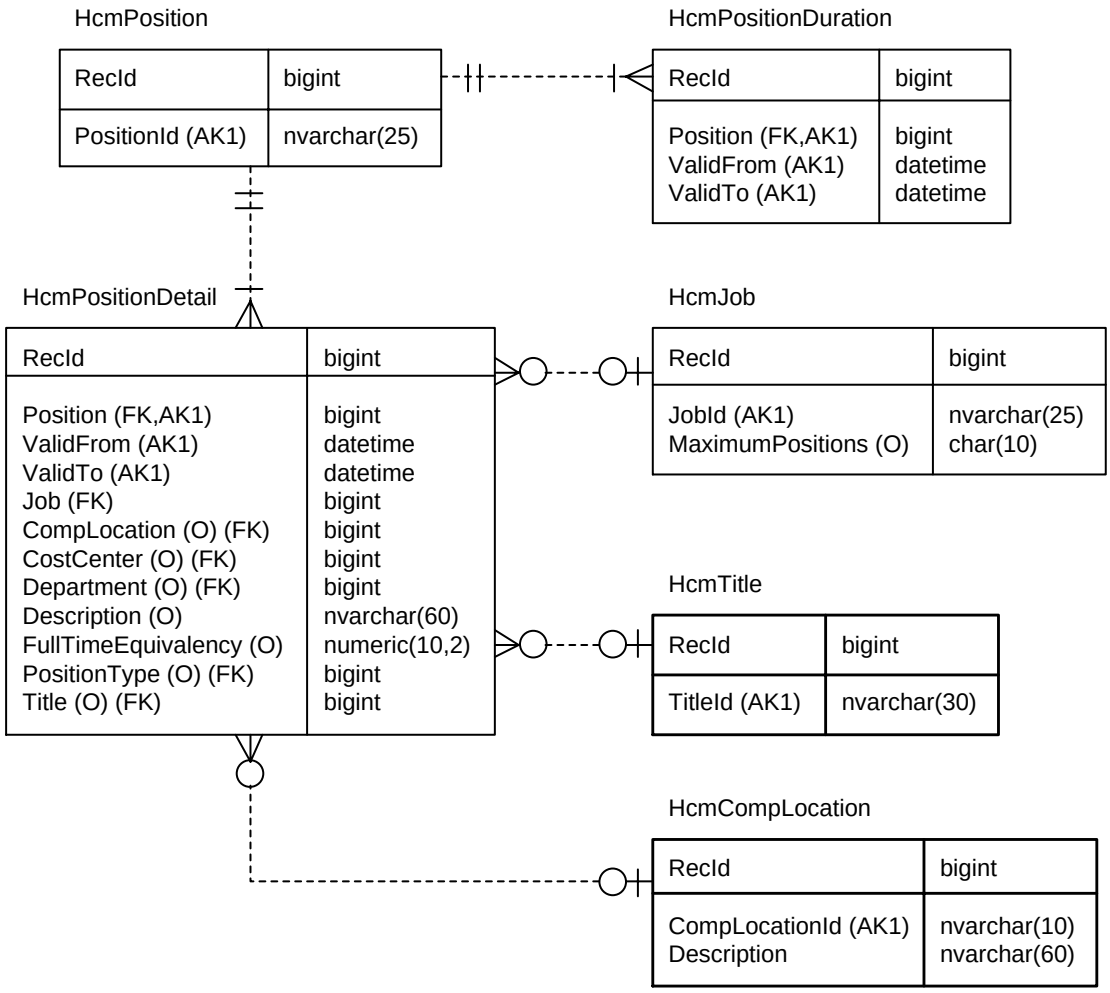
HRMLanguageCode	HcmLanguageCode
HRMLeaveType	HcmLeaveType
HRMLoanType	HcmLoanType
HRMPayrollCategory	HcmPayrollCategory
HRMPayrollDeductionType	HcmPayrollDeductionType
HRMPayrollFrame	HcmPayrollFrame
HRMPayrollPremium	HcmPayrollPremium
HRMPayrollScaleLevel	HcmPayrollScaleLevel
HRMRatingLevel	HcmRatingLevel
HRMRatingModel	HcmRatingModel
HRMReasonCode	HcmReasonCode
HRMReminderType	HcmReminderType
HRMResponsibility	HcmResponsibility
HRMSkill	HcmSkill
HRMSkillType	HcmSkillType
HRMTask	HcmJobTask
HRMUnions	HcmUnions
HRMVeteranStatus	HcmVeteranStatus

Data model diagrams

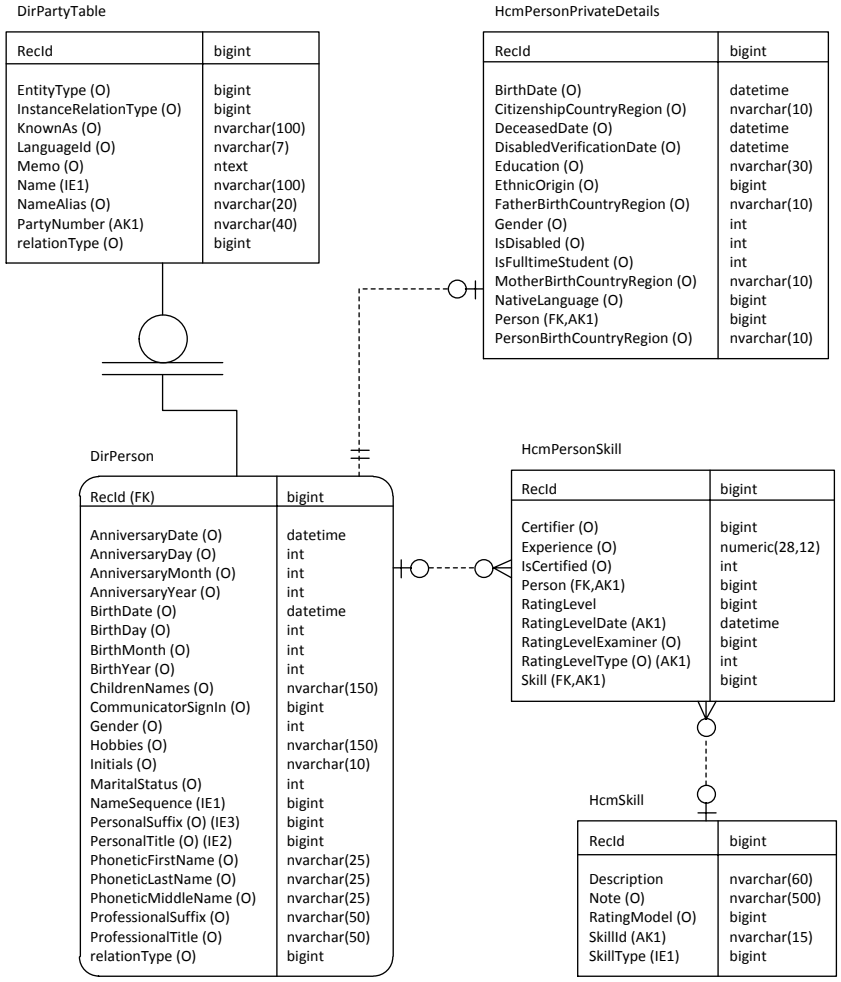
Worker (partial)



Position (partial)



Person (partial)



Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989

Worldwide +1-701-281-6500

www.microsoft.com/dynamics