

Microsoft Dynamics® AX 2012

Shared Currencies and Exchange Rates for Microsoft Dynamics AX 2012

White Paper

This document highlights the key concepts and APIs related to the calculation, display, and storage of currency and exchange rate information.

<http://microsoft.com/dynamics/ax>

Date: April 2011

Author: Paul Winje, Senior Development Lead

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



Table of Contents

Overview.....	3
Audience.....	3
Terminology	3
Currency calculations.....	4
Working with currency calculations.....	4
Calculate the accounting currency amount from a transaction currency	4
Calculate the transaction currency amount from an accounting currency.....	5
Calculate using exchange rates that have been provided	5
Calculate by overriding the default exchange rate type from the ledger	6
Calculate outside the context of a ledger.	6
Upgrading legacy currency calculation calls	7
Exchange rates	9
Retrieving exchange rates.....	9
Retrieve the exchange rates between a transaction currency and the accounting currency	9
Retrieve the exchange rates between a transaction currency and the accounting currency using static methods.....	10
Retrieve exchange rates outside the context of a ledger	10
Storing and displaying exchange rates.....	11
Display a stored exchange rate.....	11
Store an exchange rate entered by a user.....	12
Determine whether an exchange rate should be enabled on a form	12
Upgrading old exchange rate calls.....	12
Data model.....	14
Data upgrade	14

Overview

In Microsoft Dynamics® AX 2012, the currency and exchange rate framework has been enhanced to share information across multiple legal entities. As part of this work, the data model has been redesigned, the calculation engine has been rewritten, and the APIs have been updated. This document highlights the key concepts and APIs related to the calculation, display, and storage of currency and exchange rate information, and illustrates the appropriate patterns to use in application code.

Audience

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

Terminology

Microsoft Dynamics AX 2012 terms:

Term	Definition
Exchange rate	The value of one currency expressed in terms of another on a particular date.
Currency pair	The two currencies used in an exchange rate quotation.
Exchange rate type	A grouping that allows users to set up different exchange rates for a currency pair. Examples include Buy, Sell, Spot, and Budget.
Ledger	The part of an accounting system that is used for classifying the monetary value of economic transactions by using a chart of accounts, a fiscal calendar, and one or more currencies. A ledger has a one-to-one relationship with a legal entity. Transactions are performed in the context of a ledger that provides key pieces of information, such as accounting currency, reporting currency, default exchange rate type, fiscal calendar, and the chart of accounts.
Transaction currency	The currency in which a transaction originates.
Accounting currency	The primary currency in which a legal entity maintains its financial records. The accounting currency is stored in the Ledger table.
Reporting currency	The reporting currency of the ledger. The reporting currency is stored in the Ledger table. It is optional.
Default exchange rate type	An exchange rate type stored in the Ledger table that is used to specify which set of exchange rates should be used for the Ledger.

Currency calculations

Four pieces of information are necessary to perform a currency calculation in Microsoft Dynamics AX 2012:

- From currency
- To currency
- Date
- Exchange rate type

When performing a calculation in the context of a given legal entity, the exchange rate type and one of the currencies can be derived from the ledger that is associated with that legal entity. For example, assume that an accounting currency of USD and an exchange rate type of SELL have been set up for a given ledger. When that ledger is passed to the calculation engine and the **calculateTransactionToAccounting** method is called, the calculation engine is able to automatically determine that the accounting currency is USD and the exchange rate type is SELL.

In most cases, calculations will be performed in the context of a given ledger. When performing a calculation outside of the context of a ledger, it is still possible to provide all of the necessary information via **parm** methods on the calculation engine.

Working with currency calculations

The **CurrencyExchangeHelper** class was added in Microsoft Dynamics AX 2012 and is the recommended API to perform calculations between currencies. The following examples illustrate its usage for the most common scenarios.

Calculate the accounting currency amount from a transaction currency

This example calculates the amount in the context of the current ledger. This is indicated by passing *Ledger::current()* to the constructor method of the **CurrencyExchangeHelper** class.

```
CurrencyExchangeHelper currencyExchangeHelper;
TransDate transactionDate;
CurrencyCode transactionCurrency = 'CAD';
AmountCur amountToConvert = 100.50;
boolean shouldRoundResult = true;
AmountMst result;

currencyExchangeHelper = CurrencyExchangeHelper::newExchangeDate(
    Ledger::current(),
    transactionDate);

result = currencyExchangeHelper.calculateTransactionToAccounting(
    transactionCurrency,
    amountToConvert,
    shouldRoundResult);
```

Calculate the transaction currency amount from an accounting currency

This example calculates the amount in the context of a ledger other than the current ledger. This is indicated by the use of the `Ledger::primaryLedger` method.

```
CurrencyExchangeHelper currencyExchangeHelper;
TransDate transactionDate;
CurrencyCode transactionCurrency = 'CAD';
AmountMst amountToConvert = 100.50;
boolean shouldRoundResult = true;
AmountCur result;

currencyExchangeHelper = CurrencyExchangeHelper::newExchangeDate(
    Ledger::primaryLedger(CompanyInfo::findDataArea('TST').RecId),
    transactionDate);

result = currencyExchangeHelper.calculateAccountingToTransaction(
    transactionCurrency,
    amountToConvert,
    shouldRoundResult);
```

Calculate using exchange rates that have been provided

There are two important things to note in this example: (1) whenever calculations are performed, both `ExchangeRate1` and `ExchangeRate2` must always be considered due to Euro triangulation, and (2) the rates are always stored in terms of a transaction currency to the accounting currency. Therefore, if the example called the `calculateAccountToTransaction` method instead, the exchange rates should still be passed in the same order.

```
CurrencyExchangeHelper currencyExchangeHelper;
TransDate transactionDate;
CurrencyCode transactionCurrency = 'CAD';
AmountMst result;

currencyExchangeHelper = CurrencyExchangeHelper::newExchangeDate(
    Ledger::current(),
    transactionDate);

currencyExchangeHelper.parmExchangeRate1(1.234);
currencyExchangeHelper.parmExchangeRate2(2.54321);

result = currencyExchangeHelper.calculateTransactionToAccounting(
    transactionCurrency,
    543.34,
    true);
```

Calculate by overriding the default exchange rate type from the ledger

Calculating an exchange rate by overriding the default exchange rate type would be useful when it is necessary to use a different set of exchange rates for a calculation scenario. Examples might include budget processing or consolidations.

```
CurrencyExchangeHelper currencyExchangeHelper;
TransDate transactionDate;
CurrencyCode transactionCurrency = 'CAD';
AmountMst result;

currencyExchangeHelper = CurrencyExchangeHelper::newExchangeDate(
    Ledger::current(),
    transactionDate);

currencyExchangeHelper.parmExchangeRateType(
    ExchangeRateType::findByName('SpecialRateType').RecId);

result = currencyExchangeHelper.calculateTransactionToAccounting(
    transactionCurrency,
    200.75,
    true);
```

Calculate outside the context of a ledger.

Nearly every time a calculation is performed, it will be in the context of a ledger; however, there are some scenarios where a ledger might not be involved. The following example shows how to perform such a calculation.

```
CurrencyExchangeHelper currencyExchangeHelper;
TransDate transactionDate;
CurrencyCode fromCurrency = 'CAD';
CurrencyCode toCurrency = 'USD';
AmountCur result;

currencyExchangeHelper = CurrencyExchangeHelper::construct();

    currencyExchangeHelper.parmExchangeDate(transactionDate);

    currencyExchangeHelper.parmExchangeRateType(
    ExchangeRateType::findByName('SpecialRateType').RecId);

result = currencyExchangeHelper.calculateCurrencyToCurrency(
    fromCurrency,
    toCurrency,
    123.45,
    true);
```

Additional, less common scenarios are also supported. Refer to the [CurrencyExchangeHelper class documentation](#) for additional information. Always check to see whether the **CurrencyExchangeHelper** class has the method required when converting any amount in the application. We recommend that you always perform the calculations by using the engine because the engine takes all necessary factors into account.

Upgrading legacy currency calculation calls

Previously, developers calculated amounts by using the **CurrencyExchHelper** class or the Currency table methods. The **CurrencyExchHelper** class has been removed, therefore any calls referencing it will need to be refactored. Many methods on the Currency table have also been removed and will need to be refactored.

The following table documents the methods that have been removed from the Currency table along with the corresponding replacement method, where applicable.

Old method	Replacement method
editConsAvgRateNonMonetary	N/A – Exchange rate types are now used to store exchange rates specific to the consolidation process
editConsClosingRateMonetary	N/A – Exchange rate types are now used to store exchange rates specific to the consolidation process
FindExchRate	Various methods on the ExchangeRateHelper class
FindExchRateTxt	exchRateTxt method on the Currency table
isCurrencyInTriangulation	N/A – This method does not need a replacement because the triangulation flag is not used by the calculation engine any longer; if you display editable exchange rates on a form, call the isExchangeRateEditable method on the ExchangeRateHelper class to determine whether the user should be allowed to modify the rates
ledgerAccountLossName	N/A
ledgerAccountNonRealProfitLoss	N/A
ledgerAccountNonRealProfitName	N/A
ledgerAccountProfitName	N/A
priceTypeRound	roundWithRuleType method on the CurrencyExchangeHelper class
accountLoss	ledgerDimension method on the CurrencyLedgerGainLossAccount table
accountNonrealLoss	ledgerDimension method on the CurrencyLedgerGainLossAccount table
accountNonrealProfit	ledgerDimension method on the CurrencyLedgerGainLossAccount table
accountProfit	ledgerDimension method on the CurrencyLedgerGainLossAccount table
amountCur2MSTSecond	calculateTransactionToReporting method on the CurrencyExchangeHelper class
amountMST2MSTSecond	calculateAccountingToReportingAdjustment method on the CurrencyExchangeHelper class
codeCompanyCurrency	accountingCurrency method on the Ledger table
crossRate	getCrossRate method on the ExchangeRateHelper class
exchRateConsAverage	N/A – Exchange rate types are now used to store exchange rates specific to the consolidation process
exchRateConsClosing	N/A – Exchange rate types are now used to store exchange rates specific to the consolidation process
exchRateConsHistorical	N/A – Exchange rate types are now used to store exchange rates specific to the consolidation process

Old method (continued)	Replacement method (continued)
existByCompany	N/A – Currencies are now shared across all legal entities
findByCompany	N/A – Currencies are now shared across all legal entities
findExchRateSecond	getExchangeRate2 method on the ExchangeRateHelper class
roundOffAmount	roundingPrecision method on the Currency table
roundOffProject	N/A
setExchRateHelpText	N/A

The following table documents the methods that have been removed from the **CurrencyExchHelper** class along with the corresponding replacement method on the new **CurrencyExchangeHelper** class, where applicable.

Old method	Replacement method
newCurrency	newLedger or newExchangeDate
newExchDate	newExchangeDate
newExchRateHelper	N/A
calculateAmountCurToCur	calculateTransactionToTransaction
calculateAmountCurToMst	calculateTransactionToAccounting
calculateAmountCurToSecondary	calculateTransactionToReporting
calculateAmountMstToCur	calculateAccountingToTransaction
calculateAmountMstToSecondary	calculateAccountingToReportingAdjustment
parmCompany	parmLedgerRecId
parmCurrency	Now passed via method calls
parmExchDate	parmExchangeDate
parmExchRate	parmExchangeRate1
parmExchRateSecondary	parmExchangeRate2
parmIsGovernmentExchRate	Supported via exchange rate types; the parmExchangeRateTypeRecId method can be used for this
parmIsTriangulated	N/A
roundAmount	round
calculateExchRate	calcualteExchangeRate
roundAmount_Static	round

Exchange rates

The same pieces of information are necessary to work with exchange rates that are used to perform currency calculations:

- From currency
- To currency
- Date
- Exchange rate type

Retrieving exchange rates

Two exchange rates (`ExchangeRate1` and `ExchangeRate2`) must always be considered when retrieving or using exchange rates in the application. This is because Euro triangulation is possible for countries that are in the process of transitioning their national currency to the Euro currency. This results in scenarios where a currency calculation requires two exchange rates.

- Under normal circumstances, amounts could be converted directly from GBP to USD, which would require only one exchange rate:

GBP > USD

`ExchangeRate1` would be non-zero.

`ExchangeRate2` would be zero.

- Assuming that GBP is a denomination of the Euro currency, amounts would require two exchange rates (triangulation) in order to calculate GBP to USD:

GBP > EUR > USD

`ExchangeRate1` would be non-zero.

`ExchangeRate2` would be non-zero.

There are nine possible triangulation scenarios, three of which require two exchange rates. As a result, all code must assume that two exchange rates are possible and must always take this into account when storing them or passing them to the calculation engine. If a subsystem is only using one exchange rate, it is likely incorrect unless that rate is a true cross rate that factors in both exchange rates. (Note that the `getCrossRate` and `getCrossRateReciprocal` methods on the `ExchangeRateHelper` class do factor in triangulation.)

Another important concept to understand is that the rates are always stored in terms of the transaction (or reporting) currency to the accounting currency. Therefore, when calling the exchange rate engine, be sure to set the "from" and "to" currency accordingly. This is illustrated in the following examples.

The `ExchangeRateHelper` class was added in Microsoft Dynamics AX 2012 and is the recommended API to retrieve exchange rates. The following examples illustrate its usage for the most common scenarios.

Retrieve the exchange rates between a transaction currency and the accounting currency

```
ExchangeRateHelper exchangeRateHelper;  
TransDate transactionDate;  
CurrencyCode transactionCurrency = 'CAD';  
CurrencyExchangeRate exchangeRate1;  
CurrencyExchangeRate exchangeRate2;  
  
exchangeRateHelper = ExchangeRateHelper::newExchangeDate(  
    Ledger::current(),
```

```
transactionCurrency,  
transactionDate);
```

```
exchangeRate1 = exchangeRateHelper.getExchangeRate1();  
exchangeRate2 = exchangeRateHelper.getExchangeRate2();
```

Retrieve the exchange rates between a transaction currency and the accounting currency using static methods

There are performance benefits to using the instance methods, therefore those are recommended where possible.

```
ExchangeRateHelper exchangeRateHelper;  
TransDate transactionDate;  
CurrencyCode transactionCurrency = 'CAD';  
CurrencyExchangeRate exchangeRate1;  
CurrencyExchangeRate exchangeRate2;  
  
exchangeRate1 = ExchangeRateHelper::getExchangeRate1_Static(  
    Ledger::current(),  
    transactionCurrency,  
    transactionDate);  
  
exchangeRate2 = ExchangeRateHelper::getExchangeRate2_Static(  
    Ledger::current(),  
    transactionCurrency,  
    transactionDate);
```

Retrieve exchange rates outside the context of a ledger

```
ExchangeRateHelper exchangeRateHelper;  
TransDate transactionDate;  
CurrencyCode fromCurrency = 'CAD';  
CurrencyCode toCurrency = 'USD';  
CurrencyExchangeRate exchangeRate1;  
CurrencyExchangeRate exchangeRate2;  
  
exchangeRateHelper = ExchangeRateHelper::newCurrencyToCurrency(  
    fromCurrency,  
    toCurrency);  
  
exchangeRateHelper.parmExchangeDate(transactionDate);  
  
exchangeRateHelper.parmExchangeRateType(  
    ExchangeRateType::findByName('SpecialRateType').RecId);  
  
exchangeRate1 = exchangeRateHelper.getExchangeRate1();  
exchangeRate2 = exchangeRateHelper.getExchangeRate2();
```

When retrieving exchange rates in a Microsoft .NET application that integrates with Microsoft Dynamics AX 2012, or when reporting on information outside of the context of the application, there are three additional options for retrieving the exchange rates:

- Use the LedgerExchangeRateService web service call, which allows the ability to both read and write exchange rates from and to the application.
- Query the ExchangeRateEffectiveView view to retrieve the exchange rates. This is used primarily in reporting scenarios, but could also be used to gather the necessary information without calling into the application.
- Finally, the new and improved .NET Interop support added to the Microsoft MorphX®/ X++ environment makes it easier to call X++ code from .NET applications. In this case, the X++ exchange rate engine could be called directly from .NET applications.

Additional, less common scenarios are also supported. Refer to the [ExchangeRateHelper class documentation](#) for additional information. Always check to see whether the **ExchangeRateHelper** class has the method required when retrieving exchange rates in the application. We recommend that you do not retrieve exchange rates directly from the ExchangeRate table because it is likely that some key factors will be missed if you bypass the exchange rate engine.

Storing and displaying exchange rates

When storing exchange rates that have been entered by the user, you must apply the appropriate factors prior to storage. The same is true when displaying exchange rates. If the appropriate factors are not applied before storing or displaying exchange rates, incorrect rates could be used for a calculation. This is because every rate in the system is first divided by the quotation unit for the currency pair involved and then is multiplied by 100 before being stored in the database.

Display a stored exchange rate

Here is an example of how an exchange rate that has been stored in the system should be displayed. Typically, this would occur in a **display** method as shown below.

```
display ExchRate displayExchRate()
{
    ExchangeRateHelper exchangeRateHelper = ExchangeRateHelper::newCurrency(
        Ledger::current(),
        this.CurrencyCode);

    return exchangeRateHelper.displayStoredExchangeRate(this.ExchangeRate);
}
```

Store an exchange rate entered by a user

This example is similar to the previous one, but shows what to do when storing an exchange rate entered by the user. Typically, this would occur in an **edit** method as shown below.

```
public edit ExchRate editExchRate(boolean _set, ExchRate _exchRate)
{
    ExchRate exchRate = _exchRate;
    ExchangeRateHelper exchangeRateHelper = ExchangeRateHelper::newCurrency(
        Ledger::current(),
        this.CurrencyId);

    if (_set)
    {
        this.ExchRate = exchangeRateHelper.prepareExchangeRateForStorage(exchRate);
    }
    else
    {
        exchRate = exchangeRateHelper.displayStoredExchangeRate(this.ExchRate);
    }

    return exchRate;
}
```

These methods should be used any time an exchange rate is displayed on a form or report, or any time a user has the ability to enter or edit an exchange rate. Additionally, it is necessary to enable or disable the appropriate exchange rates based on the triangulation scenario. For example, because users cannot edit fixed exchange rates to the Euro currency, those exchange rates should never be available for editing.

Determine whether an exchange rate should be enabled on a form

The following example shows how to determine whether an exchange rate should be enabled on a form. Use the value returned to enable the exchange rate controls accordingly. Static methods also are available.

```
ExchangeRateHelper exchangeRateHelper;
TransDate transactionDate;
CurrencyCode transactionCurrency = 'CAD';
CurrencyExchangeRate exchangeRate1;
CurrencyExchangeRate exchangeRate2;

exchangeRateHelper = ExchangeRateHelper::newExchangeDate(
    Ledger::current(),
    transactionCurrency,
    transactionDate);

exchangeRateHelper.isExchangeRate1Editable();
exchangeRateHelper.isExchangeRate2Editable();
```

Upgrading old exchange rate calls

In previous versions of Microsoft Dynamics AX, developers retrieved exchange rates by using the **ExchRateHelper** class or the ExchRates table methods. The **ExchRateHelper** class and the ExchRates table have been removed, therefore any calls referencing them will need to be refactored.

The following table documents the methods that have been removed from the ExchRates table along with the corresponding replacement method, where applicable.

Old method	Replacement method
editExchRate	prepareExchangeRateForStorage method on the ExchangeRateHelper class; additionally, the prepareExchangeRateForStorage_Static method also is available
exchangeText	N/A
setExchRate	prepareExchangeRateForStorage method on the ExchangeRateHelper class; additionally the prepareExchangeRateForStorage_Static method is also available
showExchRate	displayStoredExchangeRate method on the ExchangeRateHelper class; additionally the displayStoredExchangeRate_Static method is also available
displayExchRate	displayStoredExchangeRate method on the ExchangeRateHelper class; additionally, the displayStoredExchangeRate_Static method is also available
exchRateCache	N/A
findExchRateDate	N/A
findExchRateDateByCompany	N/A – Exchange rates are now shared across all legal entities
flushCacheClient	N/A
flushCacheServer	N/A

The following table documents the methods that have been removed from the **ExchRateHelper** class along with the corresponding replacement method on the new **ExchangeRateHelper** class, where applicable.

Old method	Replacement method
newExchDate	newExchangeDate
getExchRate	getExchangeRate1
getExchRateSecond	getExchangeRate2
getExchRateTriangulated_Cur2Mst	getCrossRate
getExchRateTriangulated_Mst2Cur	getCrossRateReciprocal
isTriangulated	N/A – This is handled automatically and is no longer needed
parmCompany	parmLedgerRecId
parmCurrency	parmFromCurrency and parmToCurrency
parmExchDate	parmExchangeDate
parmIsGovernmentExchRate	Supported via exchange rate types; the parmExchangeRateTypeRecId method can be used for this
parmIsTriangulated	N/A – This is handled automatically and is no longer needed
getExchRate_Static	getExchangeRate1_Static

Old method (continued)	Replacement method (continued)
getExchRateSecond_Static	getExchangeRate2_Static
getExchRateTriangulated_Cur2Mst_Static	getCrossRate_Static
getExchRateTriangulated_Mst2Cur_Static	getCrossRateReciprocal_Static
isTriangulated_Static	N/A – This is handled automatically and is no longer needed

Data model

This section provides a better understanding of the new shared currency and exchange rate data model. (See Figure 1, “New currency data model” on the next page.) The mappings between the old data models and the new data models are documented in the Microsoft Dynamics AX 2012 TechNet library, in the section [Upgrade to Microsoft Dynamics AX 2012](#).

Data upgrade

Some key decisions were made in Microsoft Dynamics AX 2012 that should virtually eliminate the need for any data upgrade code for shared currencies and exchange rates:

- All foreign key references to currencies will continue to use the natural key of the Currency table and do not need to be replaced with a surrogate key.
- All exchange rates are stored exactly as they were before.
- Exchange rate types are defaulted from the ledger and do not need to be stored with transactions.

As a result of these decisions, none of the transactions referencing currencies or exchange rates requires an upgrade, unless developers have customizations that are directly related to the Currency or ExchRates tables. Refer to the Microsoft Dynamics AX 2012 TechNet library, [Upgrade to Microsoft Dynamics AX 2012](#) section for details about the logic used to upgrade all of the currency information required to perform calculations.

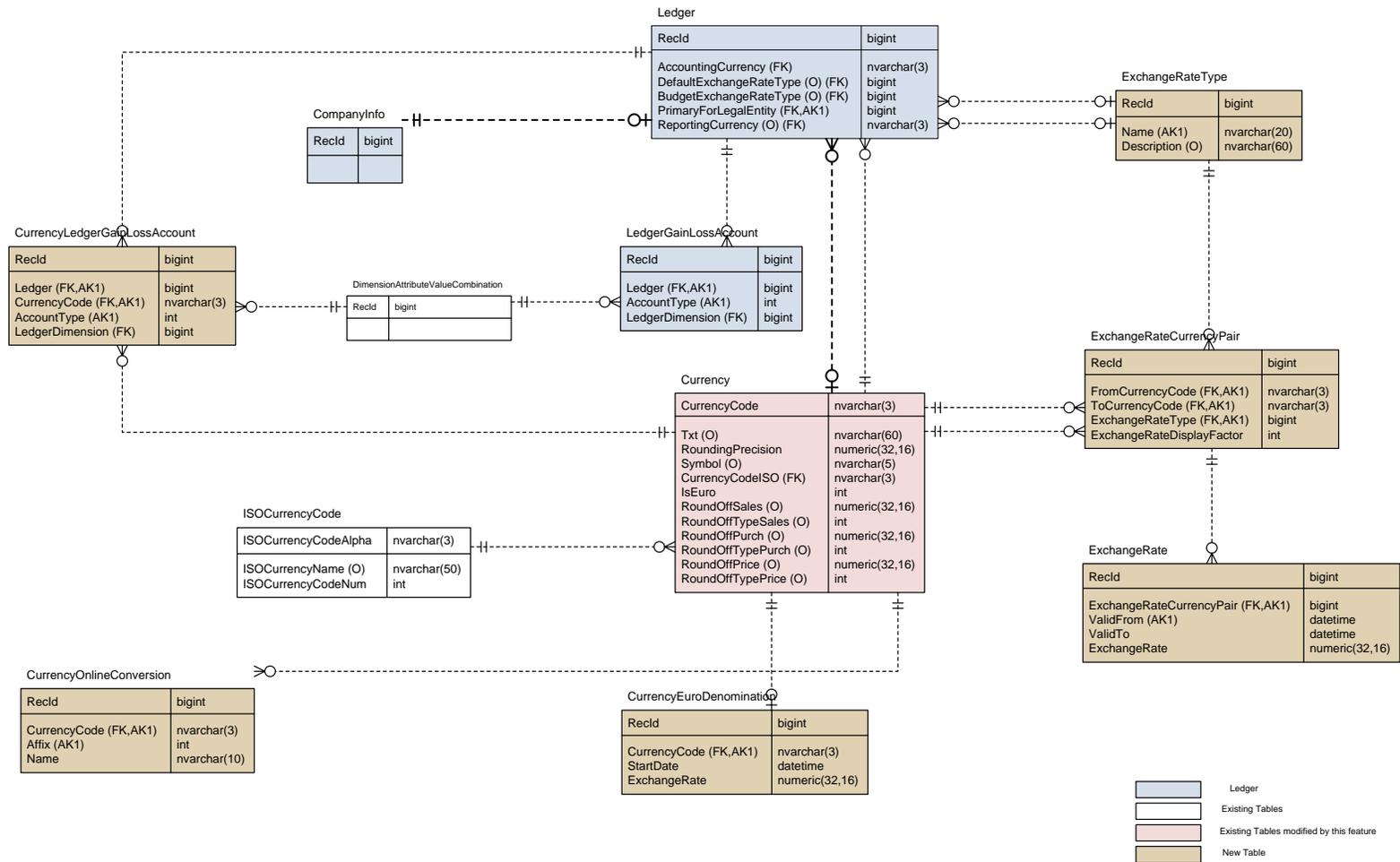


Figure 1: New currency data model

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

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes. You may modify this document for your internal, reference purposes.

© 2011 Microsoft Corporation. All rights reserved.

Microsoft, Microsoft Dynamics, the Microsoft Dynamics logo, and MorphX are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Microsoft

16