Currency Specification

New Edition: June 2000
Version 1.0

Contents

Contents

Preface

About the Object Management Group

The Object Management Group, Inc. (OMG) is an international organization supported by over 800 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization's charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG's objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based.

What is CORBA?

The Common Object Request Broker Architecture (CORBA), is the Object Management Group's answer to the need for interoperability among the rapidly proliferating number of hardware and software products available today. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them. CORBA 1.1 was introduced in 1991 by Object Management Group (OMG) and defined the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). CORBA 2.0, adopted in December of 1994, defines true interoperability by specifying how ORBs from different vendors can interoperate.

Associated OMG Documents

In addition to the other Domain Technology specifications, the CORBA documentation set includes the following:

- *Object Management Architecture Guide* defines the OMG's technical objectives and terminology and describes the conceptual models upon which OMG standards are based. It defines the umbrella architecture for the OMG standards. It also provides information about the policies and procedures of OMG, such as how standards are proposed, evaluated, and accepted.
- *CORBA: Common Object Request Broker Architecture and Specification* contains the architecture and specifications for the Object Request Broker.
- *CORBA Languages*, a collection of language mapping specifications. See the individual language mapping specifications.
- *CORBAservices: Common Object Services Specification* contains specifications for OMG's Object Services.
- *CORBAfacilities: Common Facilities Specification* includes OMG's Common Facility specifications.

The OMG collects information for each book in the documentation set by issuing Requests for Information, Requests for Proposals, and Requests for Comment and, with its membership, evaluating the responses. Specifications are adopted as standards only when representatives of the OMG membership accept them as such by vote. (The policies and procedures of the OMG are described in detail in the *Object Management Architecture Guide*.)

OMG formal documents are available from our web site in PostScript and PDF format. To obtain print-on-demand books in the documentation set or other OMG publications, contact the Object Management Group, Inc. at:

OMG Headquarters
250 First Avenue, Suite 201
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
pubs@omg.org
http://www.omg.org

0.1 Acknowledgments

The following companies submitted and/or supported parts of the CORBA Finance specification(s):

Cyborg Systems, Inc.

International Business Machines Corporation

System Software Associated, Inc.

Overview 1

Note – This specification's format has changed only - no textual changes were made since its release in December 1998.

Contents

This chapter contains the following topics.

Topic	**Page**
"Introduction"	1-1
"Compliance Points"	1-2
"Relationship to Other Specifications"	1-3

1.1 Introduction

This document specifies a set of business objects and related abstractions as a proposed standard to support international currency. It describes the objectives and business requirements for each object of the component. The business abstractions defined in this specification include:

- A currency component.
- Basic business object for currency, money, and exchange rate.
- Calculation and formatting mechanisms for the use of money.

1.2 Compliance Points

1.2.1 Mandatory and Optional Requirements Met

1.2.1.1 Currency

The currency interface accommodates both an alpha code and mnemonic for the ISO standard 4217, and a description. Currency allows for user-defined currencies in which the code and mnemonic may be meaningless. The Currency interface allows for setting monetary symbols for currencies of both major and minor (fractional) parts.

A currency can be determined as active by checking the current date or the date of exchange against the introduction and expiration date of the currency. The date will be in compliance with the final standard for date/time encompassed by the common facilities. Major and minor parts of currency are handled with a ratio defined between the major and minor currency.

1.2.1.2 Money

The internal precision of money is maintained by using the common facilities decimal amount value and by setting the precision on the money calculator.

Legal arithmetic operations of adding and subtracting two money amounts, multiplying and dividing money by a scalar value, and comparing moneys will be supported by the money calculator interface and the common money business object.

Expressions containing mixed currencies are only supported in the **FbcCurrency** module. To use mixed currencies in the **CboCurrency** module, the DMoney objects need to first be exchanged by the exchange rate so that they are of the same currency prior to doing any calculations on them. **CBOCurrency** will return exceptions on calculation methods that are invoked for monies of mixed currencies.

Both modules have interfaces defined to allow for money conversion from one currency to another and for arithmetic operations of same currency money amounts.

Monetary conversion will be handled by the exchange rate object exchange function interface and the exchange rate facility interfaces.

Money can be rounded giving the user multiple rounding rule options, such as rounding up or rounding down.

Money format externalization is handled by the money formatter interfaces. Money can be parsed from a string and formatted to a string using the money formatter interfaces to define the format structure and using the currency interfaces for currency-specific format information such as the currency symbol.

1.2.1.3 Exchange Rate

Money conversion can be done using an exchange rate. The **ExchangeRateManager** contains a set of exchange rates.

1.2.1.4 Additional

The currency abstraction allows for non-ISO (user-defined) currencies to be created. The **CurrencyBook** allows for new currencies to be registered, removed, and for a collection of all registered currencies to be accessible by an application. The **ExchangeRateManager** allows for exchange rates to be added, replaced, removed, and retrieved allowing multiple users the ability to reference the same group of exchange rates. Currency has the ability to keep track of the ISO locales (entities) associated with a specific currency. The **MoneyFormatter** interface can use the same locales for use in formatting. This will permit the formatter to determine the string representation of the money depending on the locale. Many of the interfaces use string formats for informational purposes. These fields can be set to any language and locale, however, localization of this information is outside the scope of this specification.

1.3 Relationship to Other Specifications

1.3.1 Common Business Objects and Business Object Facility

The currency objects will depend upon the Common Business Objects for date and fixed point values.

The **Currency** and **ExchangeRate** objects will depend upon a date object. The complete definition of the date object, called **DTime**, is defined in the CBO module. This Common Business Object is defined in more detail in the OMG bom/98-01-06 IBM/NIIP Revised Business Objects CBO RFP submission.

The **Money** and **ExchangeRate** objects will depend upon a decimal object. The complete definition of the decimal object, called **DDecimal**, is defined in the CBO module. This Common Business Object is defined in more detail in the OMG bom/98-01-06 IBM/NIIP Revised Business Objects CBO RFP submission.

1.3.2 Electronic Payment

Electronic payment deals with transfer of money. It would make sense for electronic payment to use the **Money** object to represent their money. The currency mnemonic aspect of the money would be whatever currency the electronic payment is dealing with.

1.3.3 Externalization Service

The Externalization Service is not used in this specification.

1.3.4 Relationship Service

The **StateIdManager** used to determine state for a network-based client depends upon the **CosObjectIdentity::IdentifiableObject** interface defined in the relationships service (OMG formal/97-12-16:CORBAservices).

1.3.5 Query Service

The **CboCurrency** module uses the **CosQueryCollection::Iterator** interface defined in the query service (OMG formal/97-12-16:CORBAservices) for the collection of currency locales.

1.3.6 Objects by Value

The specification depends upon the Objects-by-Value specification, OMG TC Document orbos/98-01-01. The **FbcCurrency** module's **Currency**, **Money**, and **ExchangeRate** are pass-by-value objects which use the value type concept where pass-by-value semantics are required. They are defined as values and are used as return types and parameters in many of the interfaces. **CboCurrency** also uses **DMoney** and **DExchangeRateDateBased** as pass-by-value objects. The **DDecimal**, **DTime**, and **DAmountOfTime** abstractions in the CBO module are also value types. The concept of initializers as defined in the Objects-by-Value specification is used in the **FbcCurrency** value objects and **CboCurrency** objects, by the keyword identifier init for the name of the operation.

1.3.7 Fit with OMA

The OMA is the Object Management Architecture. Currency fits within the "vertical domain-specific interfaces" for the Financial vertical marketplace.

Modules and Interfaces 2

Note – This specification's format has changed only - no textual changes were made since its release in December 1998.

Contents

This chapter contains the following topics.

Topic	**Page**
"Currency Modules"	2-2
"FbcCurrency Module"	2-6
"CboCurrency Module"	2-25
"CBO Module"	2-29

2.1 Introduction

This document specifies a set of business objects and related abstractions as a proposed standard to support international currency. It describes the objectives and business requirements for each object of the component. The business abstractions defined in this specification include:

- A currency component.
- Basic business object for currency, money, and exchange rate.
- Calculation and formatting mechanisms for the use of money.

2.2 Currency Modules

2.2.1 Purpose

This specification standardizes objects encompassing the concepts of currency and money. This does not limit the specification to those abstractions, but extends to other related abstractions necessary for comprehensive interfaces.

2.2.2 Overview

The currency specification uses value types as defined in the Objects-by-Value specification. These value types have minimal behavior and contain mostly accessor methods. The primary purpose of these objects is to encapsulate data. These value types are used as parameters or return types to the business utilities. The following UML class diagrams show these values as Utility classes. The diagrams show the relationships between the main currency facility abstractions. This section of the document describes the diagrams shown in Figure 2-1, Figure 2-2, and Figure 2-3.

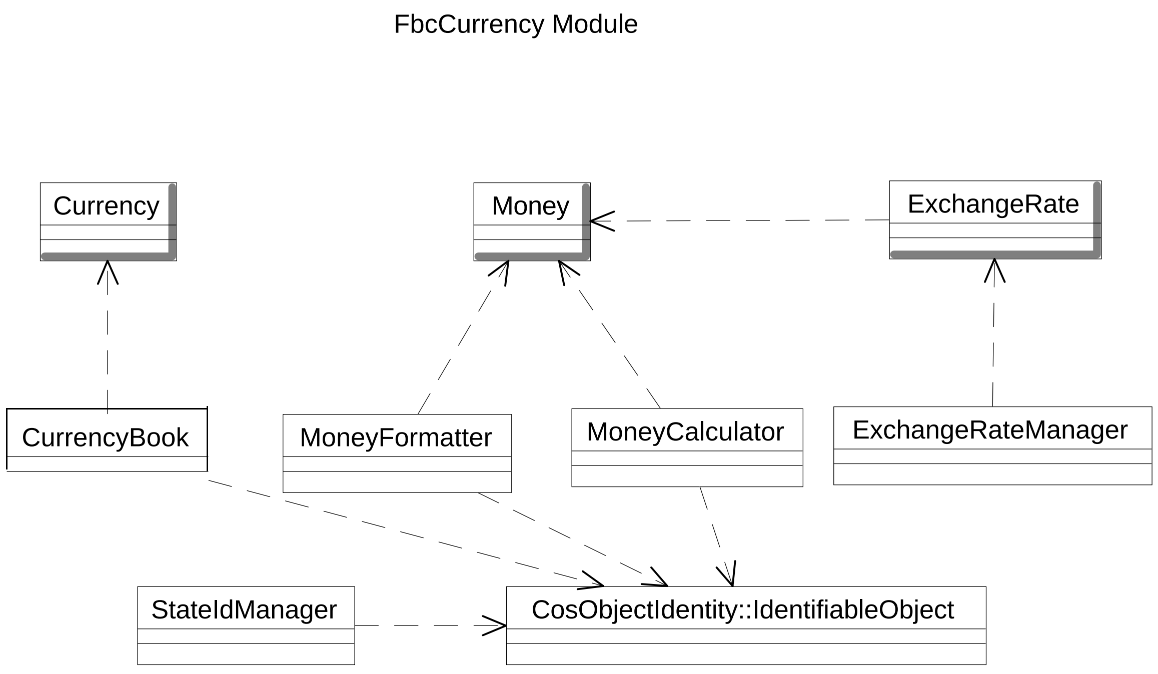

Figure 2-1 FbcCurrency Module

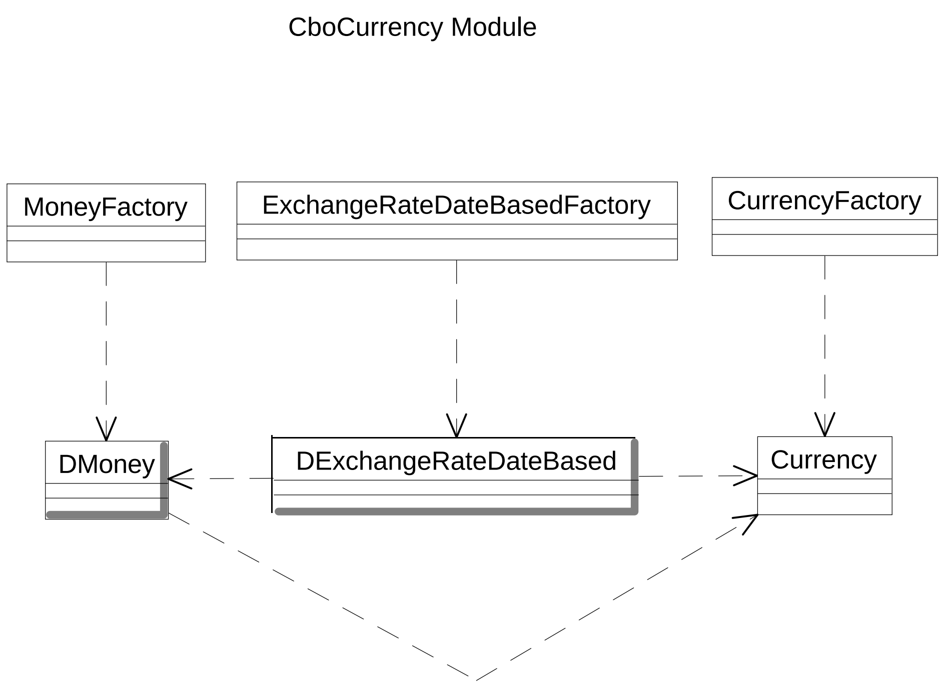

Figure 2-2 CboCurrency Module

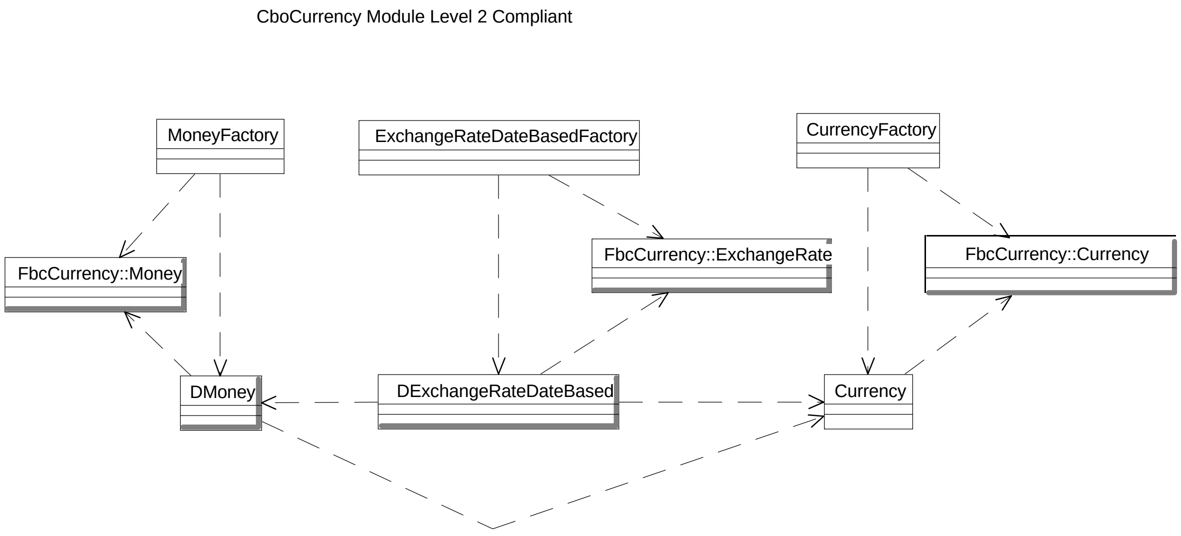

Figure 2-3 CboCurrency Module Level 2 Compliant

2.2.3 Abstractions

2.2.3.1 Network Business Component Architecture

There are abstractions for **Currency**, **Money**, **ExchangeRate**, **CurrencyBook**, **MoneyCalculator**, **MoneyFormatter**, **ExchangeRateManager**, and **StateIdManager**.

Values

The **Currency** object contains operations for setting and accessing the attributes of a particular currency.

The **Money** object contains operations for setting and accessing the amount and type of currency for a particular instance of money.

The **ExchangeRate** operations will support conversions of money from one currency to money of another in addition to setting and accessing the attributes of a particular exchange rate.

Interfaces

The **StateIdManager** is an abstraction used to identify the application client in cases where state is important. The id returned from the **StateIdManager** is used to access the other abstractions where state is necessary. The id is a **CosObjectIdentity::IdentifiableObject** defined in the relationships service specification. The StateIdManager has no direct relationship to any of the other currency abstractions.

The **CurrencyBook** maintains a group of currencies. It is used by the **MoneyFormatter** to retrieve the currency symbol and by the **MoneyCalculator** to retrieve the base currency when converting to base currency.

The **ExchangeRateManager** maintains exchange rates. It is used by the **MoneyCalculator** to retrieve an **ExchangeRate** to exchange **Money**.

The **MoneyCalculator** is a utility used for performing money arithmetic. It supports a standard set of operations for arithmetic calculations and additional state operations to support rounding rules, precision settings, and conversion rules. These state settings are saved on a per-client basis. The **MoneyCalculator** uses the **CurrencyBook** to retrieve the base currency and the **ExchangeRateManager** to retrieve an appropriate **ExchangeRate**. The **MoneyCalculator** takes **Money** as parameters.

There is a **MoneyFormatter** class utility used for parsing and formatting money into strings. The formatter is dependent upon state settings and therefore the identifier is used for all operations to identify the application client. The **MoneyFormatter** takes **Money** as parameters. The **MoneyFormatter** uses **Currency** to retrieve the symbol.

2.2.3.2 Common Business Object Architecture

There are abstractions for **Currency**, **Money**, and **ExchangeRateDateBased**. These interfaces should contain similar operations as the **Currency**, **Money**, and Exchange Rate values defined in the **FbcCurrency** module.

The **Currency** abstraction contains data attributes and accessor methods of a particular currency, as well as a method to check equivalence of currencies.

The **Money** abstraction contains data attributes, accessor methods, and arithmetic methods of one or more monies. The **Money** class references the **Currency** which defines it.

The **ExchangeRateDateBased** abstraction contains data attributes and accessor methods. The **ExchangeRate** interface uses **Money** as parameters for exchange.

2.2.4 Semantics

2.2.4.1 Network Business Component

The Distributed Business Component Architecture is intended for use as network-based business utilities which have courser-grained interfaces. These network-based objects are shared by multiple clients and therefore require re-entrant state. To reduce network traffic this architecture defines value types used as parameters in the utility objects, taking advantage of the pass-by-value semantics. This model supports the user that wants a shared, encapsulated component that is available to all applications on the network.

2.2.4.2 Common Business Objects

The Common Business Object Architecture's main intention is to be used as integrated application level business objects. This model would be used in an Object-Oriented application framework where the object is actually used internal to the application. Therefore the Common Business Objects have finer-grained interfaces and they do not require re-entrant state. These objects are not shared across processes, they are owned within an application.

2.2.5 Security

This specification provides auditing capabilities for the **CurrencyBook** and **Currency**. These capabilities include the vendor-supplied version of the **CurrencyBook** and determining whether the **CurrencyBook** integrity has been maintained. There is also the ability to determine if a currency is an ISO currency and which ISO version it is.

It is expected that the objects implemented for the currency services will utilize an OMG-compliant security service. This will provide the necessary security interoperability. With the exception of the auditing interfaces described above, there are no requirements for currency that support extending the security service in any way. Therefore, there is no need to directly address or reference security service interfaces within this specification.

2.3 FbcCurrency Module

2.3.1 Module Interfaces and Values

```
value Currency;
value Money;
value ExchangeRate;

interface StateIdManager;
interface CurrencyBook;
interface ExchangeRateManager;
interface MoneyCalculator;
interface MoneyFormatter;
```

2.3.2 Typedefs

```
typedef sequence<Currency> CurrencyCollection;
typedef sequence<wstring> StringCollection;
typedef sequence<ExchangeRate> ExchangeRateCollection;
```

The **CurrencyCollection** is used by the **CurrencyBook** to pass a list of currency instances. The **StringCollection** is used to pass a list of strings such as mnemonics and locales and is represented as a sequence of strings. The **ExchangeRateCollection** is used to pass a list of exchange rate instances.

2.3.3 Conversion Types

```
enum ConversionType { NO_CONVERSION,
            BASE_CURRENCY_CONVERSION,
            AUTOMATED_CONVERSION };
```

The conversion type is a set of options to determine what type of exchange rate conversion should be used in the **MoneyCalculator**.

- If **NO_CONVERSION** is set, then arithmetic operations performed on monies of different currencies will throw an exception.
- If **BASE_CURRENCY_CONVERSION** is set, then arithmetic operations on monies of different currencies will convert both monies to monies of the base currency as defined by the CurrencyBook. The monies are converted by retrieving the correct exchange rate from the ExchangeRateManager. The exchange rate is retrieved by the unique combination of the source and target currencies; where the source is the currency of the operands and the target is the base currency. An exception is thrown if the appropriate exchange rate cannot be found.

- If **AUTOMATED_CONVERSION** is set and arithmetic operations are performed on two monies of different currencies, then the first operand will be converted to the currency of the second operand. The exchange rate is retrieved by the unique combination of the source and target currencies; where the source is the currency of the first operand and the target is the currency of the second operand. An exception is thrown if the appropriate exchange rate cannot be found.

2.3.4 Rounding Types

```
enum RoundingType { ROUND_DOWN,
                    ROUND_UP,
                    ROUND_FLOOR,
                    ROUND_CEILING,
                    DONT_ROUND };
```

Rounding types are used by the **MoneyCalculator** to determine how rounding should be performed during arithmetic operations.

- When **ROUND_DOWN** is set, all decimal places past the internal precision will be truncated.
- When **ROUND_UP** is set, the first decimal place past the internal precision will be rounded up if greater than the rounding digit.
- When **ROUND_FLOOR** is set, a positive number will be rounded up and a negative number will be rounded down using the round up and round down rules.
- When **ROUND_CEILING** is set, a positive number will be rounded down and a negative number will be rounded up using the round down and round up rules.
- When **DONT_ROUND** is set, no rounding will occur and internal precision will be ignored.

2.3.5 Exceptions

```
enum ExceptionType { INVALID_ROUNDING_DIGIT,
                     INVALID_PRECISION,
                     AMBIGUOUS_STRING,
                     DOES_NOT_EXIST,
                     ALREADY_EXISTS,
                     INVALID_CURRENCY,
                     AMBIGUOUS_EXCHANGE_RATE,
                     UNKNOWN_LOCALE,
                     UNKNOWN_EXCEPTION };

exception FbcException
{
   ExceptionType error;
   wstring description
};
```

The error field is used to determine the type of exception that was raised. The description gives more detail to the error message.

2.3.6 Values

There are value types defined in the **FbcCurrency** which are **Currency**, **Money**, and **ExchangeRate**. Because **FbcCurrency** is a network component-based architecture passing these basic objects as values helps control network usage. These abstractions have minimal behavior and contain mostly accessor methods. The primary purpose of these objects is to encapsulate data. These value types are used as parameters or return types to the business utilities. Each value defines an initialization method as specified in the Objects-by-Value specification for Initializers. See Appendix A "Consolidated IDL Specifications" for the format of the initialization operations.

2.3.6.1 Currency Value

The currency object represents a user-defined or ISO-defined currency. It is a pass-by-value object.

value Currency

Unique Identifier Accessors

wstring getMnemonic() raises (FbcException);
void setMnemonic(in wstring mnemonic) raises (FbcException);

short getNumericCode() raises (FbcException);
void setNumericCode(in short numericCode) raises (FbcException);

wstring getName() raises (FbcException);
void setName(in wstring name) raises (FbcException);

The currency mnemonic, numeric code, and name are all unique identifiers of the currency. The ISO 4217:1995 standard defines name, numeric code, and mnemonic for each of the ISO currencies. Users creating non-ISO currencies will need to maintain the uniqueness of the name, numeric code, and mnemonic. The mnemonic is used by money to identify the currency associated with the money.

Attribute Accessors

wstring getFractionName() raises(FbcException);
void setFractionName(in wstring name) raises(FbcException);

The fractional name is the name of the fraction part of a currency. Examples of fractional names are "Cents" for the US dollar and "Pence" for British Pound.

wstring getSymbol() raises (FbcException);
void setSymbol(in wstring symbol) raises (FbcException);

wstring getFractionSymbol() raises(FbcException);
void setFractionSymbol(in wstring symbol) raises(FbcException);

The symbol of a currency is the symbol of the base value of a currency. The fractional symbol is the symbol of the fractional part of a currency. These symbols are used by the money formatter to format a money of a particular currency.

short getRatioOfFractionToWhole() raises(FbcException);
void setRatioOfFractionToWhole(in short ratio) raises(FbcException);

The ratio of the fraction part to whole part of a currency is the ratio between the smallest unit of the fractional part of a currency and one unit of the whole part of a currency. The ratio for United States Dollars is one hundred pennies to one dollar which is 100. In cases where there is no minor part of a currency (i.e., Zen), this method should return a value of 1.

CBO::DTime getIntroductionDate() raises(FbcException);
void setIntroductionDate(in CBO::DTime date) raises(FbcException);

CBO::DTime getExpirationDate() raises(FbcException);
void setExpirationDate(in CBO::DTime date) raises(FbcException);
boolean isCurrentlyActive() raises (FbcException);

The introduction date of a currency is the date in which it became a valid currency. The expiration date is the date in which the currency is no longer valid. An implementor will need to account for default expiration dates in cases where no expiration date has been set by a client, this would assume to indicate that this is an active currency.

wstring getDescription() raises(FbcException);
void setDescription(in wstring description) raises(FbcException);

Unused data for a currency is not exceptionable so vendors must return a wstring, whether it is a null value, spaces, or indicates a message. This would be the case for a currency that does not contain a minor currency. The **getFractionName()** should still return **wstring**.

ISO Currency

boolean isISOCurrency() raises(FbcException);
wstring getISOVersion() raises(FbcException);

A currency can be initialized as an ISO-defined currency. The isISO method returns a boolean to say whether the currency is ISO-defined or not. ISO-defined currencies will have the ISO Standard version number in which they are defined. There are no corresponding set methods because they can be defined only at the initialization time of a currency.

Locale Information

StringCollection getLocales() raises(FbcException);
void addLocale(in wstring locale) raises (FbcException);

void removeLocale(in wstring locale) raises (FbcException);

A currency can have a set of locales in which it is used. Valid locales are defined to correspond one to one with the currency entity lists defined in the ISO 4217:1995 standard on currency. A user can add and remove locales from any currency. If a locale is changed on an ISO currency, it will no longer be considered ISO. The getLocales method will return the set of all locales for which this currency is recognized. An exception will be thrown if a locale is used that is not contained in the list of ISO locales.

2.3.7 Money Value

Money refers to a specific amount of a particular currency. The abstraction for Money must clearly support editing and reading this amount as well as the knowledge of its currency.

```
value Money {
    CBO::DDecimal getValue() raises(FbcException);
    void setValue(in CBO::DDecimal amount) raises(FbcException);

    wstring getCurrencyMnemonic() raises(FbcException);
    void setCurrencyMnemonic(in wstring mnemonic)
    raises(FbcException);
};
```

The **Money** value is supported with a decimal abstraction that is defined as a common object defined in the CBO module.

The money definition above implies that, roughly, currency is the type (meta data) while money is the instance of the type. Therefore, money is in the context of, and must have specific knowledge of, its defining currency. To support this, the money abstraction defined in this specification supports access to its unique currency identifier.

2.3.8 ExchangeRate Value

value ExchangeRate

Mnemonic Identifiers

```
wstring getSourceCurrencyMnemonic() raises (FbcException);
void setSourceCurrencyMnemonic(in wstring currencyMnemonic)
         raises (FbcException);

wstring getTargetCurrencyMnemonic () raises(FbcException);
void setTargetCurrencyMnemonic(in wstring currencyMnemonic)
         raises (FbcException);
```

Each exchange rate has a source and target currency to identify what currencies to convert with. An exception will be raised if the exchange rate has not been initialized and the get source or target mnemonic is called.

Conversion Factor

CBO::DDecimal getConversionFactor() raises(FbcException);
void setConversionFactor(in CBO::DDecimal conversionFactor)
raises(FbcException);

There are methods to set and get the conversion factor. This conversion factor is used to do the actual conversion calculations of money amounts.

Exchange Rate Type

wstring getType() raises(FbcException);
void setType(in wstring exchangeRateType)
raises(FbcException);

There are methods to set and get the exchange rate types. A user will define the types of the exchange rates that are created. Exchange rate types are not used by any of the internal code.

Exchanging Money

Money exchange(in Money sourceMoney) raises(FbcException);

The exchange method will take the source money and convert it to the target currency defined in the exchange rate. An exception will be raised if the source money is of a different currency than the source currency in the exchange rate. The converted money will be returned.

2.3.9 State Identifier Interface

interface StateIdManager

The **StateIdManager** interface uses the **CosObjectIdentity::IdentifiableObject** interface from the Relationships Service. It identifies the network client to the currency component so that state information can be used.

CosObjectIdentity::IdentifiableObject getStateIdentifier() raises (FbcException);

The first time a network client uses the currency component, it will need to get the identifier to its state settings. This identifier will be used for all methods in the currency component which set or require state information.

void removeStateIdentifier(in CosObjectIdentity::IdentifiableObject identi-fier) raises (FbcException);

When a network client is completely done using the currency component it will release its identifier. This will free its state information.

2.3.10 CurrencyBook Interface

interface CurrencyBook

The **CurrencyBook** represents a set of ISO-defined or user-defined currencies.

CurrencyBook Identification

wstring getPublishedVersion() raises (FbcException);
boolean isIntegrityMaintained() raises (FbcException);

The published version is set by the currency component vendor. It is a unique string that identifies the version of the **CurrencyBook**.

Integrity is maintained on the **CurrencyBook** as long as none of the original currencies are changed. Integrity will not be maintained if an original currency is replaced or deleted.

Retrieving Currency Information

CurrencyCollection getAllCurrencies() raises (FbcException);
StringCollection getAllCurrencyMnemonics() raises (FbcException);
StringCollection getAllCurrencyLocales() raises (FbcException);

Getting all currencies will return a sequence of currency objects representing currencies defined in the **CurrencyBook.**

Getting all currency mnemonics will return a sequence of strings representing the currency mnemonics associated with the set of all currencies defined in the **CurrencyBook**.

Getting all currency locales will return a sequence of strings each representing a locale. The locale strings represent the set of all the ISO 4217:1995 defined entity strings representing locales for ISO currencies.

Currency Accessors

```
void addCurrency(in Currency currency)
        raises (FbcException);

void removeCurrency(in wstring currencyMnemonic)
        raises (FbcException);

void replaceCurrency(in Currency currency)
        raises (FbcException);
```

```
Currency getCurrency(in wstring mnemonic)
        raises (FbcException);

boolean containsCurrency(in wstring mnemonic)
        raises (FbcException);

boolean areEquivalent(in Currency currency,
                  in Currency comparison)
        raises (FbcException);
```

Add currency adds a currency to the **CurrencyBook**. An exception is raised if the currency already exists in the **CurrencyBook** or if the locales of the currency are not standard locales. Adding currencies will have no impact on the integrity of the **CurrencyBook**.

Remove currency removes a specific currency, identified by its unique mnemonic, from the **CurrencyBook**. Removing currencies that were provided by the vendor will change the integrity of the **CurrencyBook**. Removing currencies that were previously added will not impact the integrity of the **CurrencyBook**.

Replacing a currency requires that the mnemonic, numeric code, and name are the same for both the currency in the **CurrencyBook** and the currency passed in. Replacing a currency that was provided by the vendor will change the integrity of the **CurrencyBook**. Replacing currencies that were added will not impact the integrity of the **CurrencyBook**.

The **getCurrency** interface will return the currency object for the currency mnemonic passed in. If the currency does not exist in the **CurrencyBook**, an exception will be raised.

The **containsCurrency** interface will return true if the currency represented by the passed-in mnemonic exists in the **CurrencyBook** and false if it does not.

The **areEquivalent** interface determines whether the two passed-in currency objects contain the same elements.

Base Currency

```
void setBaseCurrencyMnemonic(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in wstring baseCurrency)
       raises (FbcException);

wstring getBaseCurrencyMnemonic(
            in CosObjectIdentity::IdentifiableObject stateIdentifer)
       raises (FbcException);
```

The **setBaseCurrency** interface passes in a state identifier. This preserves the state information for the particular application using the **CurrencyBook** component. If no currency matches the passed-in mnemonic, an exception is raised. The base currency is used by the **MoneyFormatter**.

The **getBaseCurrencyMnemonic** interface returns the mnemonic of the base currency. An exception is raised if no base currency has been set. The state identifier is passed in so that the **CurrencyBook** can determine which client needs this state information.

2.3.11 Exchange Rate Manager Interface

interface ExchangeRateManager

The exchange rate manager contains a set of exchange rates and manages the adding and removing of exchange rates. The exchange rate manager is also used by the money calculator to retrieve exchange rates for the conversion of money during arithmetic operations. The exchange rate manager contains no state information. The combination of source currency, target currency, and rate type must be unique among the exchange rates in the exchange rate manager.

Creating Exchange Rates

```
ExchangeRate createExchangeRate(
    in wstring rateTypeId,
    in wstring sourceCurrencyMnemonic,
    in wstring targetCurrencyMnemonic,
    in CBO::DDecimal conversionFactor)
    raises(FbcException);
```

Exchange rates are created by passing in the attribute data. The **ExchangeRate** is returned to the user.

Exchange Rate Methods

```
void addExchangeRate(in ExchangeRate exchangeRate)
     raises(FbcException);

void removeExchangeRate(in ExchangeRate exchangeRate)
     raises(FbcException);

void replaceExchangeRate(in ExchangeRate exchangeRate)
     raises(FbcException);
```

Adding an exchange rate will add the exchange rate to the exchange rate manager. An exception will be raised if the exchange rate already exists in the manager.

Removing an exchange rate will remove the exchange rate from the exchange rate manager if a match of the source currency, target currency, and rate type is found. An exception will be raised if a matching exchange rate cannot be found.

Replacing an exchange rate will replace the conversion rate in the exchange rate specified, if a matching exchange rate is found in the exchange rate manager. An exception will be raised if no matching exchange rate is found to be replaced.

Retrieval Methods

```
ExchangeRate getExchangeRateForRateType(
                        in wstring rateTypeId,
                        in wstring sourceCurrencyMnemonic,
                        in wstring targetCurrencyMnemonic)
      raises(FbcException);

ExchangeRate getExchangeRate(
                        in wstring sourceCurrencyMnemonic,
                        in wstring targetCurrencyMnemonic)
      raises(FbcException);
```

The **getExchangeRateForRateType** operation will return an exchange rate object if the requested exchange rate is found. An exception is raised if no matching exchange rate is found. An exception is raised from the **getExchangeRate** operation if there are multiple matching exchange rates for the source and target mnemonics.

2.3.12 Money Formatter

The money formatter is used to format and parse money. Since many applications can use it at once, there is a state identifier to determine which formatting settings to use.

2.3.12.1 Default Symbols for Money Format String

The Money Formatter contains default symbols for specifying formatting strings. The symbols are:

Symbol	Meaning
0	placeholder for a digit
#	placeholder for a digit, zero shows as absent
.	placeholder for a radix symbol
,	placeholder for a grouping symbol
*	placeholder for a currency symbol
@	placeholder for a currency fraction symbol
M	placeholder for a currency mnemonic
;	separate negative format from positive format
-	placeholder for negative prefix

- The radix character is the delimiter between the whole and fraction parts of a money amount.
- The grouping symbol is the delimiter between groups of digits of the whole part of a money amount.
- The currency symbol, currency fraction symbol, and currency mnemonic may all be used together or only one or two can be used in the same string.

2.3.12.2 User-Defined Symbols Operations

```
wchar getPatternCurrencySymbol(
            in CosObjectIdentity::IdentifiableObject stateIdentifier)
            raises(FbcException);
void  setPatternCurrencySymbol(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in wchar patternCurrencySymbol)
            raises(FbcException);

wchar getPatternFractionSymbol(
            in CosObjectIdentity::IdentifiableObject stateIdentifier)
            raises(FbcException);
void   setPatternFractionSymbol(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in wchar patternFractionSymbol)
            raises(FbcException);

wchar getPatternMnemonicSymbol(
            in CosObjectIdentity::IdentifiableObject stateIdentifier)
            raises(FbcException);
void   setPatternMnenonicSymbol(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in wchar patternMnemonicSymbol)
            raises(FbcException);

wchar  getPatternDigit(
            in CosObjectIdentity::IdentifiableObject stateIdentifier)
            raises(FbcException);
void   setPatternDigit(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in wchar patternDigit)
            raises(FbcException);

wchar  getPatternRadix(
            in CosObjectIdentity::IdentifiableObject stateIdentifier)
            raises(CBOException);
void   setPatternRadix(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in wchar patternRadix)
            raises(FbcException);
```

Each of the **setPattern** and **getPattern** operations sets or gets the pattern symbol for the format string. This allows the users to use symbols that make sense to them if the default symbols do not make sense. A **stateidentifier** is passed in to each of these interfaces because the symbol patterns are specific to each application using the Currency component.

2.3.12.3 Formatter Specification Operations

```
void setFormatByLocale(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in wstring locale)
        raises (FbcException);
```

The formatter contains default formatting information for each locale supported by the ISO 4217:1995 standard. This interface will set the money formatter to use that default locale format. An exception will be raised if the locale is not one of the ISO locales.

```
wstring getFormattingString(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);
void setFormattingString(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in wstring formattingString)
        raises (FbcException);
```

There are operations to get and set the formatting string. The formatting string determines the format of the money amount. The format is passed in as "positive string; negative string." The user can define what symbols to use for each piece of the formatting string or use the vendor-provided defaults.

```
wstring getRadixCharacter(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);
void setRadixCharacter(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in wstring radixCharacter)
        raises (FbcException);

wstring getGroupingSymbol(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);
void setGroupingSymbol(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in wstring groupingSymbol)
        raises (FbcException);
```

The radix character is the delimiter between the whole and fraction part of the money amount. The grouping symbol is the delimiter between groups of digits of the whole part of a money amount. The pattern settings show where these symbols should be. The actual symbol must also be set.

```
short getInputMulitplier(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);
```

```
void setInputMultiplier(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in short multiplier)
        raises (FbcException);

short getOutputDivisor(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);
void setOutputDivisor(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in short divisor)
        raises (FbcException);
```

The input multiplier and output divisor are for shifting the decimal place. This is a base 10 amount, so a multiplier of 1 shifts over one place which is the same as multiplying by 10.

2.3.12.4 *Format and Parse Operations*

```
wstring format(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in Money money)
        raises (FbcException);

Money parse(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in wstring moneyString)
        raises (FbcException);

Money parseForCurrency(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in wstring moneyString,
        in wstring currencyMnemonic)
        raises (FbcException);
```

The format interface formats a money amount returning a formatted string. The format uses the currency of the money to determine the appropriate currency symbol to use. The formatter uses the **CurrencyBook** to retrieve the currency symbol information. If the currency is not found in the **CurrencyBook**, an exception will be raised.

The parse interface takes a string and parses it into a money structure. An exception will be raised if the string is ambiguous, such as if the symbol or mnemonic is not in the list of currencies in the **CurrencyBook** or if the symbol is in multiple currencies in the **CurrencyBook**.

The parse for currency by mnemonic parses the string using the currency specified. An exception will be raised when the currency mnemonic cannot be found in the **CurrencyBook**.

2.3.13 Money Calculator Interface

The money calculator abstraction is defined as a stateless calculator with no operand context. This means that both operands must be supplied before binary operations can occur. The money calculator does contain state for precision, rounding rules, and conversion rules. Operations which affect the state of the money calculator or need to know the state of the money calculator to perform the operation must be passed the application state identifier.

2.3.13.1 Money Calculator Attributes

```
double getInternalPrecision(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);

void setInternalPrecision(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in double precision)
        raises (FbcException);
```

The internal precision determines the decimal precision of the money amount. If the rounding type is **DONT_ROUND**, the internal precision is ignored. Otherwise, all rounding rules are used at the specified precision.

```
RoundingType getRounding(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
          raises (FbcException);

void setRounding(
         in CosObjectIdentity::IdentifiableObject stateIdentifier,
         in RoundingType roundingFlag)
         raises (FbcException);

short getRoundingDigit(
         in CosObjectIdentity::IdentifiableObject stateIdentifier)
         raises (FbcException);

void setRoundingDigit(
         in CosObjectIdentity::IdentifiableObject stateIdentifier,
         in short roundingDigit)
         raises (FbcException);
```

The rounding type is used to determine what type of rounding to use for arithmetic operations. **RoundingType** is a set of enumerations, each described in the enumeration section of this document.

The rounding digit is used to determine the value of the internal precision digit on which the rounding rules take place. Valid rounding digits are 1 through 9. An example of the **RoundingDigit** is:

```
Rounding Digit=4
Internal Precision=2
Value= .383   Rounded= .38
Value= .384   Rounded= .39

ConversionType getConversion(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);

void setConversion(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in ConversionType type)
        raises (FbcException);
```

The **ConversionType** is used to determine what type of conversion to use for arithmetic operations on monies of different currencies. **ConversionType** is a set of enumerations, each described in the enumeration section of this document.

The **CurrencyBook** is used to determine the base currency when the **ConversionType** is convert to base.

The **ExchangeRateManager** is used to determine what exchange rate to use when it is necessary to exchange money.

2.3.13.2 Arithmetic Operations

```
Money add(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in Money oper1,
        in Money oper2)
        raises (FbcException);

Money subtract(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
          in Money oper1,
          in Money oper2)
          raises (FbcException);

Money multiply(
          in CosObjectIdentity::IdentifiableObject stateIdentifier,
          in CBO::DDecimal multiplier,
          in Money money)
          raises (FbcException);

Money divide(
          in CosObjectIdentity::IdentifiableObject stateIdentifier,
          in CBO::DDecimal divisor,
          in Money dividend)
          raises (FbcException);
```

```
Money abs(in Money oper1)
        raises (FbcException);

Money round(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in Money oper1)
        raises (FbcException);
```

The **MoneyCalculator** allows addition of two money operands, returning the result. The **MoneyCalculator** allows subtraction of two money operands, subtracting the second operand from the first operand and returning the result.

Addition and subtraction is allowed on monies of different currency types if the conversion type is set to either base currency conversion or automatic conversion. Base currency conversion uses the **CurrencyBook** to determine the base currency. Both conversion types use the **ExchangeRateManager** to exchange the money. Automatic conversion uses the currency of the first operand as the source currency and the currency of the second operand as the target currency for the exchange, returning money of the target currency. Exceptions will be raised if the appropriate exchange rate is not found or if there is not a unique source/target exchange rate to use.

Money multiplication returns the resulting money of a money multiplied by a scalar value. Money division returns the resulting money of a money divided by a scalar value.

The **MoneyCalculator** can determine the absolute value of a money operand, returning the resulting money.

The **MoneyCalculator** will round a money operand, returning the resulting money. Rounding uses the rounding and internal precision setting in the calculator.

2.3.13.3 Relational Operations

```
boolean lessThan(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in Money oper1,
        in Money oper2)
        raises (FbcException);

boolean equal(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in Money oper1,
        in Money oper2)
        raises (FbcException);

boolean greaterThan(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in Money oper1,
        in Money oper2)
        raises (FbcException);
```

Relational comparisons can be performed on monies of the same currency type or monies of differing currency types. When a comparison is done on money operands of differing currency types, the conversion type is used to determine whether to raise an exception or to convert the money to a common currency type prior to the comparison. Base currency conversion uses the **CurrencyBook** to determine the base currency. Both conversion types use the **ExchangeRateManager** to exchange the money. Automatic conversion uses the currency of the first operand as the source currency and the currency of the second operand as the target currency for the exchange, returning money of the target currency. Exceptions will be raised if the appropriate exchange rate is not found or if there is not a unique source/target exchange rate to use.

2.3.14 Examples of Use

2.3.14.1 Currency Calculations with Mixed Currency Using the FbcCurrency Components

Refer to Figure 2-4 on page 2-23 for an illustration of currency calculations with mixed currency using the FbcCurrency components. Figure 2-5 on page 2-24 illustrates using the FbcCurrency::MoneyFormatter.

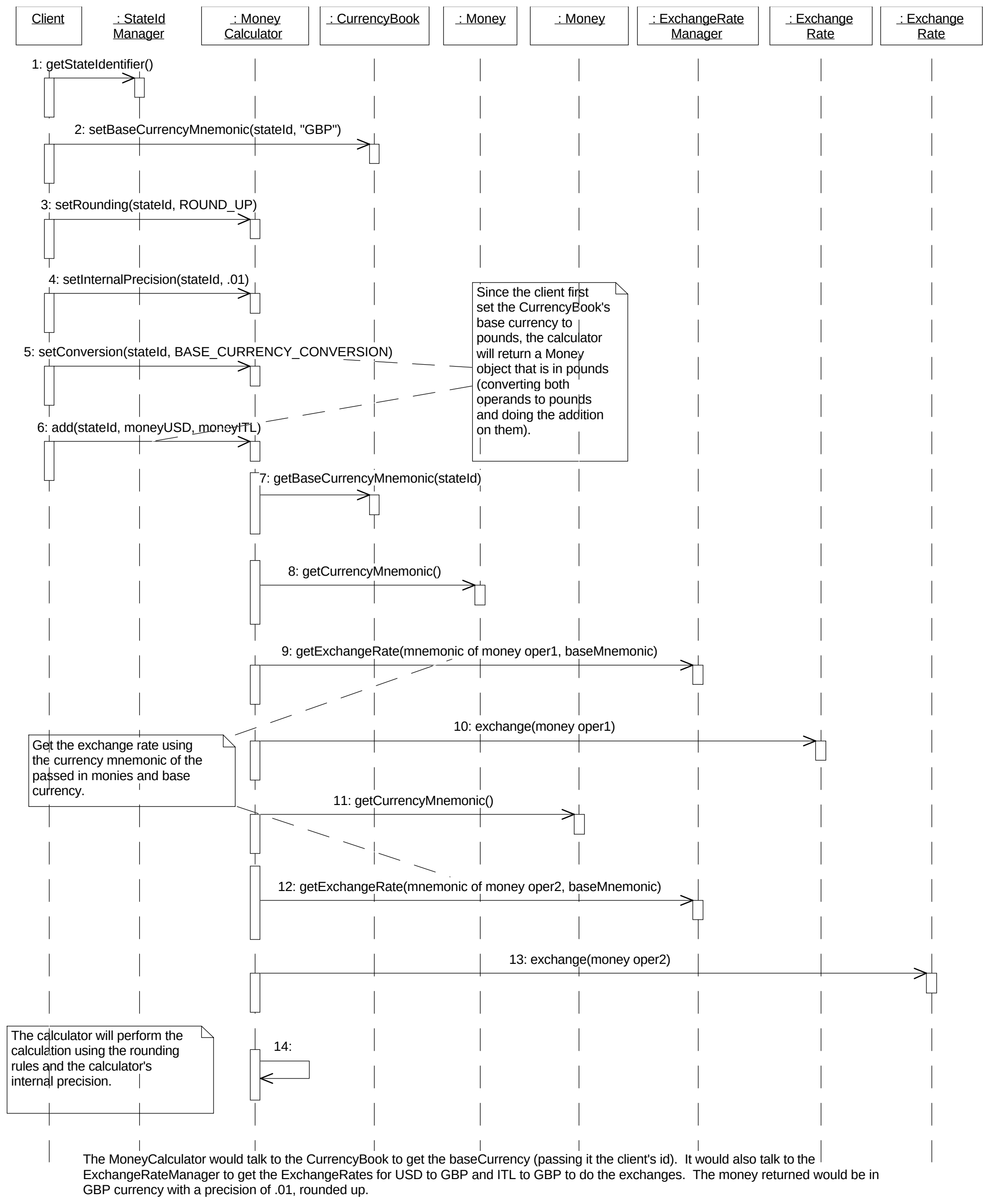
Client
: StateId Manager
: Money Calculator
: CurrencyBook
: Money
: Money
: ExchangeRate Manager
: Exchange Rate
: Exchange Rate
1: getStateIdentifier()
2: setBaseCurrencyMnemonic(stateId, "GBP")
3: setRounding(stateId, ROUND_UP)
4: setInternalPrecision(stateId, .01)
5: setConversion(stateId, BASE_CURRENCY_CONVERSION)
6: add(stateId, moneyUSD, moneyITL)
Since the client first set the CurrencyBook's base currency to pounds, the calculator will return a Money object that is in pounds (converting both operands to pounds and doing the addition on them).
7: getBaseCurrencyMnemonic(stateId)
8: getCurrencyMnemonic()
9: getExchangeRate(mnemonic of money oper1, baseMnemonic)
10: exchange(money oper1)
Get the exchange rate using the currency mnemonic of the passed in monies and base currency.
11: getCurrencyMnemonic()
12: getExchangeRate(mnemonic of money oper2, baseMnemonic)
13: exchange(money oper2)
The calculator will perform the calculation using the rounding rules and the calculator's internal precision.
14:
The MoneyCalculator would talk to the CurrencyBook to get the baseCurrency (passing it the client's id). It would also talk to the ExchangeRateManager to get the ExchangeRates for USD to GBP and ITL to GBP to do the exchanges. The money returned would be in GBP currency with a precision of .01, rounded up.

Figure 2-4 Currency Calculations with Mixed Currency Using the FbcCurrency Components

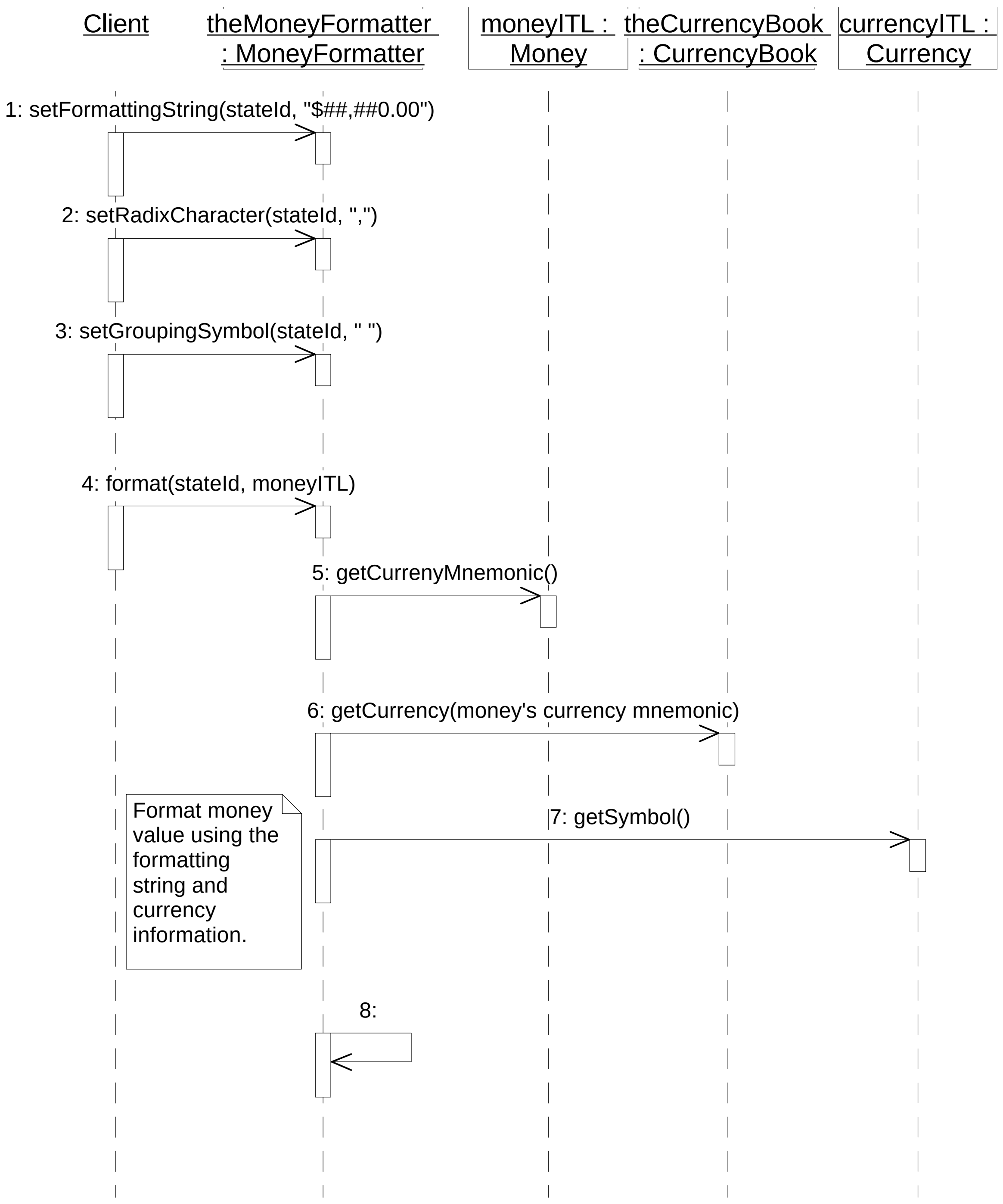
Client
theMoneyFormatter : MoneyFormatter
moneyITL : Money
theCurrencyBook : CurrencyBook
currencyITL : Currency
1: setFormattingString(stateId, "$##,##0.00")
2: setRadixCharacter(stateId, ",")
3: setGroupingSymbol(stateId, " ")
4: format(stateId, moneyITL)
5: getCurrenyMnemonic()
6: getCurrency(money's currency mnemonic)
7: getSymbol()
Format money value using the formatting string and currency information.
8:

Figure 2-5 Using the FbcCurrency::MoneyFormatter

2.4 CboCurrency Module

2.4.1 Module Interfaces and Values

Each interface and value contain an initialization method which will allow implementors to adapt to various protocols such as Enterprise Beans, the methods identified in the Objects-by-Value specification, or the use of factories in a Business Object Facility.

```
interface Currency;
value DMoney;
value DExchangeRateDateBased;
```

2.4.2 Currency Interface

interface Currency

The **CboCurrency::Currency** interface defines similar accessor methods as the **FbcCurrency::Currency** interface. Only the additional interface methods will be described here.

2.4.2.1 Initialization

```
void initializeCurrency( in wstring name,
                  in wstring mnemonic,
                  in wstring symbol,
                  in long scaleFactor,
                  in short ratioOfFractionToWhole,
                  in wstring description,
                  in CBO::DTime introductionDate,
                  in CBO::DTime expirationDate,
                  in boolean isISO,
                  in boolean isExternalOutputShowingFractions,
                  in boolean isInternalOutputShowingFractions)
       raises(CBO::CBOException);
```

The initialization method is used to initialize a currency.

2.4.2.2 Money Formatting Attributes

```
boolean isExternalOutputShowingFractions() raises(CBO::CBOException);
void setExternalOutputShowingFractions(in boolean flag)
    raises(CBO::CBOException);

boolean isInternalOutputShowingFractions() raises(CBO::CBOException);
void setInternalOutputShowingFractions(in boolean flag)
    raises(CBO::CBOException);
```

The booleans for the internal and external output showing fractions are available for use by applications to allow different users in an application to control and select different options for the display of the currency values. The applications control the setting and use of these attributes.

2.4.2.3 *Locale Information*

CosQueryCollection::Iterator getLocaleIterator()
raises(CBO::CBOException);

The **getLocaleIterator()** method will return a **CosQueryCollection::Iterator** that can be used to access the locales defined for a **Currency**.

2.4.2.4 *Equality Testing*

boolean equals(in Object another Currency)
raises(CBO::CBOException);

The equals method will check the values of the objects, not instance equality.

2.4.3 *Money*

value DMoney

The **CboCurrency::DMoney** value defines similar accessor methods as **FbcCurrency::Money**. Only the additional interface methods will be described here.

2.4.3.1 *Initialization and Getting/Setting Data*

```
void initializeDMoney(in Currency theCurrency,
            in CBO::DDecimal amount) raises(CBO::CBOException);
```

The initialize interface allows the Money to be initialized with the Currency and the amount.

```
Currency getCurrency() raised(CBO::CBOException);
void setCurrency(in Currency theCurrency)
    raises(CBO::CBOException);
```

The **Currency** object is used in these operations instead of the identifying currency mnemonic. This allows the user to retrieve or set the actual currency that this money is defined by.

2.4.3.2 *Money Arithmetic*

```
DMoney add(in DMoney anotherMoney) raises(CBO::CBOException);
DMoney subtract(in DMoney anotherMoney) raises(CBO::CBOException);
```

DMoney multiply(in CBO::DDecimal multiplier) raises(CBO::CBOException);
DMoney divide(in CBO::DDecimal denominator) raises(CBO::CBOException);

The add method adds two monies of like currencies together and returns a new instance of money containing the result. An exception is raised if the monies are of differing currencies.

The subtract method subtracts the passed-in money from the money which the method is called. The subtract method returns a new instance of money containing the result. An exception is raised if the monies are of differing currencies.

The multiply method multiplies the passed-in scalar value with the money the method is called on. A new money instance is returned.

The divide method divides the money amount by the passed-in scalar value. A new money instance is returned. An exception is raised if the passed-in scalar value is zero.

2.4.3.3 Money Logical Operations

```
boolean equals(in Object anotherMoney) raises(CBO::CBOException);
boolean greater(in DMoney anotherMoney) raises(CBO::CBOException);
boolean greaterEqual(in DMoney anotherMoney)
        raises(CBO::CBOException);
boolean greaterThanZero() raises(CBO::CBOException);
boolean less(in DMoney anotherMoney) raises(CBO::CBOException);
boolean lessEqual(in DMoney anotherMoney)
        raises(CBO::CBOException);
boolean lessThanZero() raises(CBO::CBOException);
boolean isZero() raises(CBO::CBOException);
boolean isOfSameCurrency(in DMoney anotherMoney)
        raises(CBO::CBOExcep tion);
```

Each of the logical operations takes the current **DMoney** object and does the comparison against the passed-in **DMoney** object. As an example, greater will compare this Money greater than another money.

The equals method will check the values of the objects, not instance equality. It will check if the monies are of the same currency and same amount.

2.4.3.4 Money Rounding and Truncation

DMoney createRounded(in long places) raises(CBO::CBOException);
DMoney createTruncated(in long places) raises(CBO::CBOException);

The **createRounded** and **createTruncated** operations will create a new **DMoney** instance. The new money instance will contain a rounded or truncated amount from the original money instance. The truncation or round will occur at the passed-in places digit.

DMoney createCeiling() raises(CBO::CBOException);
DMoney createFloor() raises(CBO::CBOException);

The **createFloor** and **createCeiling** operations will return new **DMoney** instances. The new money instances will have the amount changed to be rounded up or down depending upon whether they are positive or negative numbers. The ceiling of a positive number is the amount rounded up, the ceiling of a negative number is the amount rounded down. The floor is just the opposite.

```
DMoney createAbs() raises(CBO::CBOException);
DMoney createChangeSign() raises(CBO::CBOException);
```

The **createAbs** method returns an instance of **DMoney** with the amount as the absolute value of the money amount being operated on.

The **createChangeSign** returns a new instance of **DMoney** with the sign of the amount reversed from the money amount being operated on.

2.4.4 Exchange Rate Date Based

value DExchangeRateDateBased

The **CboCurrency::ExchangeRateDateBased** value defines similar operations as the **FbcCurrency::ExchangeRate**. Only the additional interface methods will be described here.

```
void  initializeDExchangeRateDateBased( in wstring rateTypeID,
                  in Currency fromCurrency,
                  in Currency toCurrency,
                  in CBO::DTime startDate,
                  in CBO::DTime endDate,
                  in CBO::DDecimal conversionFactor)
            raises(CBO::CBOException);

Currency   getSourceCurrency() raises (CBO::CBOException);
void   setSourceCurrency(in Currency newSourceCurrency)
            raises(CBO::CBOException);

Currency   getTargetCurrency() raises (CBO::CBOException);
void   setTargetCurrency(in Currency newTargetCurrency)
            raises(CBO::CBOException);

DMoney exchange(in DMoney money)
            raises(CBO::CBOException);
```

The exchange rate can exchange the money passed in. An exception will be raised if the source currency in the money being exchanged did not match the source currency in the exchange rate.

2.4.4.1 Exchange Rate Dates

```
CBO::DTime getStartDate() raises(CBO::CBOException);
void setStartDate(in CBO::DTime startDate) raises(CBO::CBOException);
```

CBO::DTime getEndDate() raises(CBO::CBOException);
void setEndDate(in CBO::DTime endDate) raises(CBO::CBOException);

Exchange rates have start and end dates in which they are valid.

2.4.4.2 Equality Testing

boolean equals(in Object another ExchangeRate)
raises(CBO::CBOException);

The equals method will check the values of the objects, not instance equality.

2.5 CBO Module

The Common Business Object module defined in this specification is a subset of the CBO module defined in the Joint Common Business Object Revised Submission. Refer to the document: OMG bom/98-01-06 IBM/NIIIP Revised Business Objects CBO RFP submission for more details and for information on each of the interfaces. The CBO idl is listed and specified in Appendix A "Consolidated IDL Specifications." The following is a general overview of the Common Business Objects abstractions used by the Currency facility.

2.5.1 CBO Values and Interfaces

value DDecimal;
value DAmountOfTime;
value DTime;

interface DDecimalFactory;
interface DAmountOfTimeFactory;
interface DTimeFactory;

2.5.2 DDecimal

DDecimal is a common business object which provides the capabilities required to set, retrieve, and perform mathematical operations with and upon a decimal value with a specified precision. **DDecimal** allows a decimal value to be used in collections and as a property value. The **DDecimal** interface provides functionality such as rounding and truncation.

2.5.3 DAmountOfTime

A **DAmountOfTime** object represents an absolute (positive) amount of time that can be added to, subtracted from, or used to hold differences of **DTime** objects. A **DAmountOfTime** object is defined as a specified number of weeks, days, hours, minutes, and seconds.

2.5.4 DTime

A **DTime** object represents an actual date and time. A **DTime** object encapsulates a point in time defined by a calendar date (year, month, day and an hour, minute, and second in that day). Formatting date and time information according to a specified locale is supported by the **DTime** interface.

2.5.5 DDecimalFactory

The **DDecimalFactory** is used to create **DDecimal** objects.

2.5.6 DAmountOfTimeFactory

The **DAmountOfTimeFactory** is used to create **DAmountOfTime** objects.

2.5.7 DTimeFactory

The **DTimeFactory** is used to create **DTime** objects.

Consolidated IDL Specifications A

The following idl specifications depend on the CORBA specification for the wstring and wchar types. The idl also depends on the pass-by-value semantics. It uses idl from the **CosQueryCollection** module of the Query Service and idl from the CosObjectIdentity module of the Relationships Service. The Object interface used in the **CboCurrency** and CBO modules defined in CORBA.idl must also be available. Because many of these specifications are not yet available, the submitters have compiled this idl by:

- using typedefs on string and char to wstring and wchar,
- changing value types to interface types, and
- dummying any other non-available service types in the appropriate included idl file.

A.1 Network Business Component IDL

```
#ifndef _FbcCurrency_idl_
#define _FbcCurrency_idl_

#include <ObjectIdentity.idl>
#include <CBO.idl>

module FbcCurrency {
    value Currency;
    value Money;
    value ExchangeRate;

    interface StateIdManager;
    interface CurrencyBook;
    interface ExchangeRateManager;
    interface MoneyFormatter;
    interface MoneyCalculator;

    typedef sequence<Currency> CurrencyCollection;
typedef sequence<ExchangeRate> ExchangeRateCollection;
typedef sequence<string> StringCollection;

enum ConversionType { NO_CONVERSION,
                      BASE_CURRENCY_CONVERSION,
                      AUTOMATED_CONVERSION };

enum RoundingType { ROUND_DOWN,
                    ROUND_UP,
                    ROUND_FLOOR,
                    ROUND_CEILING,
                    DONT_ROUND };

enum ExceptionType { INVALID_ROUNDING_DIGIT,
                   INVALID_PRECISION,
                   AMBIGUOUS_STRING,
                   DOES_NOT_EXIST,
                   ALREADY_EXISTS,
                   INVALID_CURRENCY,
                   AMBIGUOUS_EXCHANGE_RATE,
                   UNKNOWN_LOCALE,
                   UNKNOWN_EXCEPTION };

// Exceptions
exception FbcException {
   ExceptionType error;
   wstring description;
};


value Currency
{
   void   init(
                    in wstring name,
                    in wstring mnemonic,
                    in wstring numericCode,
                    in wstring symbol,
                    in wstring fractionSymbol,
                    in long scaleFactor,
                    in short ratioOfFractionToWhole,
                    in wstring description,
                    in CBO::DTime introductionDate,
                    in CBO::DTime expirationDate,
                    in boolean isISO,
                    in wstring ISOVersion,
                    in boolean isExternalOutputShowingFractions,
                    in boolean isinternalOutputShowingFractions)
          raises(FbcException);

   wstring getMnemonic() raises (FbcException);
   void setMnemonic(in wstring mne) raises (FbcException);

   short getNumericCode() raises (FbcException);
   void setNumericCode(in short numericCode) raises (FbcException);
```

```
    wstring getName() raises (FbcException);
    void setName(in wstring name) raises (FbcException);

    wstring getFractionName() raises (FbcException);
    void setFractionName(in wstring name) raises (FbcException);

    wstring getSymbol() raises (FbcException);
    void setSymbol(in wstring symbol) raises (FbcException);

    wstring getFractionSymbol() raises (FbcException);
    void setFractionSymbol(in wstring symbol) raises (FbcException);

    short getRatioOfFractionToWhole() raises (FbcException);
    void setRatioOfFractionToWhole(in short ratio)
                  raises (FbcException);

    CBO::DTime getIntroductionDate() raises (FbcException);
    void setIntroductionDate(in CBO::DTime date) raises (FbcException);

    CBO::DTime getExpirationDate() raises (FbcException);
    void setExpirationDate(in CBO::DTime date) raises (FbcException);
                      boolean isCurrentlyActive() raises (FbcException);

                      wstring getDescription() raises (FbcException);
    void setDescription(in wstring description) raises (FbcException);

    boolean isISOCurrency() raises (FbcException);
    wstring getISOVersion() raises (FbcException);

    StringCollection getLocales() raises (FbcException);
    void addLocale(in wstring locale)
      raises (FbcException); // UNKNOWN_LOCALE

    void removeLocale(in wstring locale)
        raises (FbcException); // UNKNOWN_LOCALE
  };

  value Money
  {
    void   init(in wstring currencyMnemonic,
              in CBO::DDecimal theValue)
               raises(FbcException);

    wstring getCurrencyMnemonic() raises (FbcException);
    void setCurrencyMnemonic(in wstring currencyMnemonic)
      raises (FbcException);

    CBO::DDecimal getValue() raises (FbcException);
    void setValue(in CBO::DDecimal amount) raises (FbcException);
  };
value ExchangeRate
  {
    void  init( in wstring rateTypeID,
                   in wstring sourceCurrencyMnemonic,
                   in wstring targetCurrencyMnemonic,
                    in CBO::DDecimal conversionFactor)
               raises(FbcException);

      wstring getSourceCurrencyMnemonic() raises (FbcException);
      void setSourceCurrencyMnemonic(in wstring currencyMnemonic)
               raises(FbcException);

      wstring getTargetCurrencyMnemonic () raises(FbcException);
      void setTargetCurrencyMnemonic(in wstring currencyMnemonic)
               raises(FbcException);

      CBO::DDecimal getConversionFactor() raises(FbcException);
      void setConversionFactor(in CBO::DDecimal conversionFactor)
               raises(FbcException);

      wstring getType() raises(FbcException);
      void setType(in wstring exchangeRateType)
               raises(FbcException);

      Money exchange(in Money sourceMoney) raises(FbcException);
   };

   interface StateIdManager
   {
      CosObjectIdentity::IdentifiableObject getStateIdentifier() raises (FbcException);
      void removeStateIdentifier(in CosObjectIdentity::IdentifiableObject identifier)
raises(FbcException);
   };

   interface CurrencyBook
   {

      wstring getPublishedVersion() raises (FbcException);
      boolean isIntegrityMaintained() raises (FbcException);

      CurrencyCollection getAllCurrencies() raises (FbcException);
      StringCollection getAllCurrencyMnemonics() raises (FbcException);
      StringCollection getAllCurrencyLocales() raises (FbcException);

      void addCurrency(in Currency currency)
            raises (FbcException); // ALREADY_EXISTS
      void removeCurrency(in wstring mnemonic)
            raises (FbcException); // DOES_NOT_EXIST
      void replaceCurrency(in Currency currency)
            raises (FbcException); // DOES_NOT_EXIST

      Currency getCurrency(in wstring mnemonic)
            raises (FbcException); // DOES_NOT_EXIST

      boolean containsCurrency(in wstring mnemonic)
               raises(FbcException);

      void setBaseCurrencyMnemonic(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in wstring baseCurrencyMnemonic)
```

```
        raises(FbcException); // DOES_NOT_EXIST

    wstring getBaseCurrencyMnemonic(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises(FbcException);

    boolean areEquivalent(in Currency currency,
                    in Currency comparison)
        raises (FbcException);
};


interface MoneyCalculator
{

    double getInternalPrecision(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);
    void setInternalPrecision(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in double internalPrecision)
        raises (FbcException); // INVALID_PRECISION

    RoundingType getRounding(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);
    void setRounding(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in RoundingType roundingFlag)
        raises (FbcException);

    short getRoundingDigit(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);
    void setRoundingDigit(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in short roundingDigit)
        raises (FbcException); // INVALID_ROUNDING_DIGIT

    ConversionType getConversion(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);

    void setConversion(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in ConversionType type) raises (FbcException);

    Money add(in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in Money oper1,
        in Money oper2)
        raises (FbcException);

    Money subtract(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in Money oper1,
            in Money oper2)
            raises (FbcException);

        Money multiply(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in CBO::DDecimal multiplier,
            in Money money)
            raises (FbcException);

        Money divide(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in CBO::DDecimal divisor,
            in Money dividend)
            raises (FbcException);

        Money abs(in Money oper1)
            raises (FbcException);

        Money round(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in Money oper1)
            raises (FbcException);

        boolean lessThan(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in Money oper1,
            in Money oper2)
            raises (FbcException);

        boolean equal(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in Money oper1,
            in Money oper2)
            raises (FbcException);

        boolean greaterThan(
            in CosObjectIdentity::IdentifiableObject stateIdentifier,
            in Money oper1,
            in Money oper2)
            raises (FbcException);
    };


    interface MoneyFormatter
    {
        void setFormatByLocale(
           in CosObjectIdentity::IdentifiableObject stateIdentifier,
           in wstring locale)
           raises (FbcException); // UNKNOWN_LOCALE

        wchar getPatternCurrencySymbol(
           in CosObjectIdentity::IdentifiableObject stateIdentifier)
           raises(FbcException);
        void  setPatternCurrencySymbol(
           in CosObjectIdentity::IdentifiableObject stateIdentifier,
    in wchar patternCurrencySymbol)
    raises(FbcException);

wchar  getPatternFractionSymbol(
    in CosObjectIdentity::IdentifiableObject stateIdentifier)
    raises(FbcException);
void   setPatternFractionSymbol(
    in CosObjectIdentity::IdentifiableObject stateIdentifier,
    in wchar patternFractionSymbol)
    raises(FbcException);

wchar  getPatternMnemonicSymbol(
    in CosObjectIdentity::IdentifiableObject stateIdentifier)
    raises(FbcException);
void   setPatternMnenonicSymbol(
    in CosObjectIdentity::IdentifiableObject stateIdentifier,
    in wchar patternMnemonicSymbol)
    raises(FbcException);

wchar  getPatternDigit(
    in CosObjectIdentity::IdentifiableObject stateIdentifier)
    raises(FbcException);
void   setPatternDigit(
    in CosObjectIdentity::IdentifiableObject stateIdentifier,
    in wchar patternDigit)
      raises(FbcException);

wchar  getPatternRadix(
    in CosObjectIdentity::IdentifiableObject stateIdentifier)
    raises(FbcException);
void   setPatternRadix(
    in CosObjectIdentity::IdentifiableObject stateIdentifier,
    in wchar patternRadix)
    raises(FbcException);

wstring getFormattingString(
    in CosObjectIdentity::IdentifiableObject stateIdentifier)
    raises (FbcException);
void setFormattingString(
    in CosObjectIdentity::IdentifiableObject stateIdentifier,
    in wstring formattingString)
    raises (FbcException); // AMBIGUOUS_STRING

wstring getRadixCharacter(
    in CosObjectIdentity::IdentifiableObject stateIdentifier)
    raises (FbcException);
void setRadixCharacter(
    in CosObjectIdentity::IdentifiableObject stateIdentifier,
    in wstring radixCharacter)
    raises (FbcException);

wstring getGroupingSymbol(
    in CosObjectIdentity::IdentifiableObject stateIdentifier)
    raises (FbcException);
void setGroupingSymbol(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in wstring groupingSymbol)
        raises (FbcException);

    short getInputMulitplier(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);
    void setInputMultiplier(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in short multiplier)
        raises (FbcException);

    short getOutputDivisor(
        in CosObjectIdentity::IdentifiableObject stateIdentifier)
        raises (FbcException);
    void setOutputDivisor(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in short divisor)
        raises (FbcException);

    wstring format(
        in CosObjectIdentity::IdentifiableObject stateIdentifier,
        in Money money)
        raises (FbcException);

    Money parse(
          in CosObjectIdentity::IdentifiableObject stateIdentifier,
          in wstring moneyString)
          raises (FbcException); // AMBIGUOUS_STRING

    Money parseForCurrency(
          in CosObjectIdentity::IdentifiableObject stateIdentifier,
          in wstring moneyString,
          in wstring currencyMnemonic)
          raises (FbcException);
};

interface ExchangeRateManager
{
    ExchangeRate createExchangeRate(
                            in wstring rateTypeID,
                            in wstring sourceCurrencyMnemonic,
                            in wstring targetCurrencyMnemonic,
                            in CBO::DDecimal conversionFactor)
        raises (FbcException);


    void addExchangeRate(in ExchangeRate exchangeRate)
        raises(FbcException); // ALREADY_EXISTS

    void removeExchangeRate(in ExchangeRate exchangeRate)
        raises(FbcException); // DOES_NOT_EXIST

    boolean replaceExchangeRate(in ExchangeRate exchangeRate)
        raises(FbcException); // DOES_NOT_EXIST
```

```
     ExchangeRate getExchangeRate(
                          in wstring sourceCurrencyMnemonic,
                          in wstring targetCurrencyMnemonic)
        raises (FbcException);

    ExchangeRate getExchangeRateForRateType(
                          in wstring rateTypeID,
                          in wstring sourceCurrencyMnemonic,
                          in wstring targetCurrencyMnemonic)
        raises (FbcException); // DOES_NOT_EXIST

    ExchangeRateCollection getAllExchangeRates() raises (FbcException);
  };

}; // End of FbcCurrency Module
#endif //_FbcCurrency_idl
```

A.2 Common Business Object Architecture IDL

```
#ifndef _CboCurrency_idl_
#define _CboCurrency_idl_

#include <CBO.idl>
#include <CORBA.idl>
#include <CosQueryCollection.idl>

module CboCurrency {
  // exceptions
  // exception CBOException; is picked up from CBO.IDL

  //  Interface Definitions
  interface Currency;
  value DMoney;
  value DExchangeRateDateBased;

  interface Currency
  {
    void   init(
                    in wstring name,
                    in wstring mnemonic,
                    in wstring numericCode,
                    in wstring symbol,
                    in wstring fractionSymbol,
                    in long scaleFactor,
                    in short ratioOfFractionToWhole,
                    in wstring description,
                    in CBO::DTime introductionDate,
                    in CBO::DTime expirationDate,
                    in boolean isISO,
                    in wstring ISOVersion,
                    in boolean isExternalOutputShowingFractions,
                    in boolean isinternalOutputShowingFractions)
        raises(CBO::CBOException);
```

```
long  getScaleFactor() raises(CBO::CBOException);
void   setScaleFactor(in long scaleFactor) raises(CBO::CBOException);

boolean isExternalOutputShowingFractions() raises(CBO::CBOException);
void   setExternalOutputShowingFractions(in boolean flag)
           raises(CBO::CBOException);

boolean isInternalOutputShowingFractions() raises(CBO::CBOException);
void   setInternalOutputShowingFractions(in boolean flag)
           raises(CBO::CBOException);

boolean equals(in Object anotherCurrency) raises(CBO::CBOException);

wstring getMnemonic() raises (CBO::CBOException);
void setMnemonic(in wstring mne) raises (CBO::CBOException);

short getNumericCode() raises (CBO::CBOException);
void setNumericCode(in short numericCode) raises (CBO::CBOException);

wstring getName() raises (CBO::CBOException);
void setName(in wstring name) raises (CBO::CBOException);

wstring getFractionName() raises (CBO::CBOException);
void setFractionName(in wstring name) raises (CBO::CBOException);

wstring getSymbol() raises (CBO::CBOException);
void setSymbol(in wstring symbol) raises (CBO::CBOException);

wstring getFractionSymbol() raises (CBO::CBOException);
void setFractionSymbol(in wstring symbol) raises (CBO::CBOException);

short getRatioOfFractionToWhole() raises (CBO::CBOException);
void setRatioOfFractionToWhole(in short ratio)
                        raises (CBO::CBOException);

CBO::DTime getIntroductionDate() raises (CBO::CBOException);
void setIntroductionDate(in CBO::DTime date) raises (CBO::CBOExcep tion);

CBO::DTime getExpirationDate() raises (CBO::CBOException);
void setExpirationDate(in CBO::DTime date) raises (CBO::CBOException);

wstring getDescription() raises (CBO::CBOException);
void setDescription(in wstring description) raises (CBO::CBOException);

boolean isISOCurrency() raises (CBO::CBOException);
wstring getISOVersion() raises (CBO::CBOException);

CosQueryCollection::Iterator getLocaleIterator() raises (CBO::CBOException);
void addLocale(in wstring locale)
  raises (CBO::CBOException); // UNKNOWN_LOCALE

void removeLocale(in wstring locale)
    raises (CBO::CBOException); // UNKNOWN_LOCALE
```

```
};

value DMoney
{
   // Initialization and Setting/Getting Data
   //
   void   init(in Currency theCurrency,
                    in CBO::DDecimal theValue)
             raises(CBO::CBOException);

   Currency   getCurrency() raises(CBO::CBOException);
   void setCurrency(in Currency newCurrency) raises(CBO::CBOException);

   long   getScaleFactor() raises(CBO::CBOException);

   wstring getCurrencyMnemonic() raises (CBO::CBOException);

   CBO::DDecimal getValue() raises (CBO::CBOException);
   void setValue(in CBO::DDecimal amount) raises (CBO::CBOException);

   // DMoney Arithmetic
   //
   DMoney add(in DMoney anotherMoney) raises(CBO::CBOException);
   DMoney subtract(in DMoney anotherMoney) raises(CBO::CBOException);
   DMoney multiply(in CBO::DDecimal multiplier) raises(CBO::CBOException);
   DMoney divide(in CBO::DDecimal denominator) raises(CBO::CBOException);

   // DMoney Logical Operations
   //
   boolean equals(in Object anotherMoney) raises(CBO::CBOException);
   boolean greater(in DMoney anotherMoney) raises(CBO::CBOException);
   boolean greaterEqual(in DMoney anotherMoney)
          raises(CBO::CBOException);
   boolean greaterThanZero() raises(CBO::CBOException);
   boolean less(in DMoney anotherMoney) raises(CBO::CBOException);
   boolean lessEqual(in DMoney anotherMoney)
          raises(CBO::CBOException);
   boolean lessThanZero() raises(CBO::CBOException);
   boolean isZero() raises(CBO::CBOException);
   boolean isOfSameCurrency(in DMoney anotherMoney)
          raises(CBO::CBOException);

   // DMoney Rounding and Truncation etc.
   //
   DMoney createRounded(in long places) raises(CBO::CBOException);
   DMoney createTruncated(in long places) raises(CBO::CBOException);
   DMoney createCeiling() raises(CBO::CBOException);
   DMoney createFloor() raises(CBO::CBOException);
   DMoney createAbs() raises(CBO::CBOException);
   DMoney createChangeSign() raises(CBO::CBOException);


};

value DExchangeRateDateBased
```

```
{
   void  init( in wstring rateTypeID,
                    in Currency fromCurrency,
                    in Currency toCurrency,
                    in CBO::DTime startDate,
                    in CBO::DTime endDate,
                    in CBO::DDecimal conversionFactor)
          raises(CBO::CBOException);

   Currency   getSourceCurrency() raises (CBO::CBOException);
   void   setSourceCurrency(in Currency newSourceCurrency)
          raises(CBO::CBOException);

   Currency   getTargetCurrency() raises (CBO::CBOException);
   void   setTargetCurrency(in Currency newTargetCurrency)
          raises(CBO::CBOException);

   CBO::DTime getStartDate() raises (CBO::CBOException);
   void setStartDate(in CBO::DTime startDate)
          raises(CBO::CBOException);

   CBO::DTime getEndDate() raises (CBO::CBOException);
   void setEndDate(in CBO::DTime endDate)
          raises(CBO::CBOException);

   wstring getSourceCurrencyMnemonic() raises (CBO::CBOException);
   void setSourceCurrencyMnemonic(in wstring currencyMnemonic)
          raises(CBO::CBOException);

   wstring getTargetCurrencyMnemonic () raises(CBO::CBOException);
   void setTargetCurrencyMnemonic(in wstring currencyMnemonic)
          raises(CBO::CBOException);

   CBO::DDecimal getConversionFactor() raises(CBO::CBOException);
   void setConversionFactor(in CBO::DDecimal conversionFactor)
          raises(CBO::CBOException);

   wstring getType() raises(CBO::CBOException);
   void setType(in wstring exchangeRateType)
          raises(CBO::CBOException);
                        boolean equals(in Object another ExchangeRate)
                                 raises(CBO::CBOException);

   DMoney exchange(in DMoney sourceMoney) raises(CBO::CBOException);

};

interface CurrencyFactory
{
    Currency createCurrency(in wstring name,
                      in wstring mnemonic,
                      in wstring numericCode,
                      in wstring symbol,
                      in wstring fractionSymbol,
                      in long scaleFactor,
                        in short ratioOfFractionToWhole,
                        in wstring description,
                        in CBO::DTime introductionDate,
                        in CBO::DTime expirationDate,
                        in boolean isISO,
                        in wstring ISOVersion,
                        in boolean isExternalOutputShowingFractions,
                        in boolean isInternalOutputShowingFractions)
                   raises(CBO::CBOException);


  };

  interface MoneyFactory
  {
     DMoney   createDMoney(in Currency theCurrency,
                      in CBO::DDecimal theValue) raises(CBO::CBOException);

  };


  interface  ExchangeRateDateBasedFactory
  {
       DExchangeRateDateBased createDExchangeRateDateBased(
                   in wstring rateTypeID,
                   in Currency fromCurrency,
                   in Currency toCurrency,
                   in CBO::DTime startDate,
                   in CBO::DTime endDate,
                   in CBO::DDecimal conversionFactor)
                   raises (CBO::CBOException);


   };

}; // end of CboCurrency Module
#endif // CboCurrency_idl
```

A.3 Common Business Object Module

```
#ifndef _CBO_idl_
#define _CBO_idl_

#include <CORBA.idl>

module CBO
{
  // exceptions
  exception CBOException {};

  // Interface Definitions

   value DDecimal;
   value DAmountOfTime;
  value DTime;

  interface DDecimalFactory;
  interface DAmountOfTimeFactory;
  interface DTimeFactory;


value DDecimal
 {
  DDecimal   add (in DDecimal summand) raises (CBOException);
  DDecimal   subtract(in DDecimal subtrahend) raises(CBOException);
  DDecimal   multiply(in DDecimal factor) raises(CBOException);
  DDecimal   divide(in DDecimal denominator) raises(CBOException);
  boolean   equals(in Object anObject) raises(CBOException);
  boolean   greater(in DDecimal aDecimal) raises(CBOException);
  boolean   greaterEqual(in DDecimal aDecimal) raises(CBOException);
  boolean   greaterThanZero() raises(CBOException);
  boolean   less(in DDecimal aDecimal) raises(CBOException);
  boolean   lessEqual (in DDecimal aDecimal) raises (CBOException);
  boolean   lessThanZero () raises (CBOException);
  boolean   isZero () raises (CBOException);
  boolean   isOne () raises (CBOException);
  void   setEqual (in Object dependentObject) raises (CBOException);
  void   assign (in DDecimal aDecimal)  raises (CBOException);
  void   setToZero () raises (CBOException);
  void   truncate(in long newScaleFactor) raises(CBOException);
  void   round(in long newScaleFactor) raises(CBOException);
  void   changeSignOf () raises (CBOException);
  DDecimal   changeSign() raises (CBOException);
  DDecimal   abs () raises (CBOException);
  double   getCompareValue() raises(CBOException);
  long   scaleFactor() raises (CBOException);
  void   initializeDDecimal(in wstring value, in long scaleFactor)  raises
(CBOException);
 };


interface DDecimalFactory
 {
  DDecimal   createDDecimal(in wstring value, in long scaleFactor)
raises(CBOException);
 };


 value DAmountOfTime
 {
  long   getSeconds () raises (CBOException);
  long   getMinutes () raises (CBOException);
  long   getHours () raises (CBOException);
  long   getDays () raises (CBOException);
  long   getWeeks() raises (CBOException);
  void   setSeconds (in long newSeconds) raises (CBOException);
  void   setMinutes (in long newMinutes) raises (CBOException);
  void   setHours (in long newHours) raises (CBOException);
  void   setDays (in long newDays) raises (CBOException);
  void   setWeeks(in long newWeeks) raises (CBOException);
  long   long toSeconds () raises (CBOException);
  long   toMinutes () raises (CBOException);
  long   toHours () raises (CBOException);
  long   toDays () raises (CBOException);
  long   toWeeks () raises (CBOException);
  DAmountOfTime   addTo (in DAmountOfTime anotherAmount) raises (CBOExcep-
tion);
  DAmountOfTime   difference (in DAmountOfTime anotherAmount)  raises
(CBOException);
  boolean   lessThan (in DAmountOfTime anotherAmount)  raises (CBOException);
  boolean   greaterThan (in DAmountOfTime anotherAmount)  raises
(CBOException);
  void   initializeDAmountOfTime (in long weeks,
                         in long days,
                         in long hours,
                         in long minutes,
                         in long seconds) raises (CBOException);
 };


interface DAmountOfTimeFactory
 {
  DAmountOfTime  createDAmountOfTime (in long weeks,
                                 in long days,
                                 in long hours,
                                 in long minutes,
                                 in long seconds) raises (CBOException);
 };


value DTime
 {
  long   getHours() raises (CBOException);
  long   getMinutes() raises (CBOException);
  long   getSeconds() raises (CBOException);
  long   getDayOfMonth () raises (CBOException);
  long   getDayOfYear () raises (CBOException);
  long   getMonth () raises (CBOException);
  long   getOffsetGMT () raises (CBOException);
  DAmountOfTime  getPrecision () raises(CBOException);
  void   setHours(in long hours) raises (CBOException);
  void   setMinutes(in long minutes) raises (CBOException);
  void   setSeconds(in long seconds) raises (CBOException);
  void   setDayOfMonth (in long dayOfMonth) raises (CBOException);
  void   setDayOfYear (in long dayOfYear) raises (CBOException);
  void   setMonth (in long month) raises (CBOException);
  void   setYear (in long year) raises (CBOException);
  wstring   formatDate (in wstring locale) raises (CBOException);
  wstring   formatTime (in wstring locale) raises (CBOException);
  DTime   addTo (in DAmountOfTime amount) raises (CBOException);
  DTime   subtractFrom (in DAmountOfTime amount) raises (CBOException);
  DAmountOfTime   difference (in DTime anotherTime) raises (CBOException);
  void   initializeDTime (in long year,
                   in long month,
                      in long day,
                      in long hour,
                      in long minute,
                      in long second,
                      in long offsetGMT,
                      in DAmountOfTime precision) raises (CBOException);
 };


interface DTimeFactory
 {
  DTime  createDTime(in long year,
                     in long month,
                     in long day,
                     in long hour,
                     in long minute,
                     in long second,
                     in long offsetGMT,
                     in DAmountOfTime precision) raises (CBOException);
 };

};

#endif // _CBO_idl
```

Additional Operations B

B.1 Overview

This appendix contains the additional operations needed in the Common Base Objects module to be level 2 compliant, supporting both the Network Business Component and the Common Business Objects architectures. The **FbcCurrency** module does not change; however, a few operations are added to the **CboCurrency** module. Because of this, only the changed **CboCurrency** module is shown here. The **FbcCurrency** module stays as is defined in the Consolidated IDL Specifications section. The parameters and return types defined in **FbcCurrency** return the value types defined within it. Therefore, in order to use the network components with the **CboCurrency** objects, there needs to be ways to convert the **CboCurrency** objects into **FbcCurrency** objects and vice-versa. Each factory in **CboCurrency** has an operation to create its associated type using an **FbcCurrency** object as a parameter. Each common object in **CboCurrency** has an operation defined to create the associated **FbcCurrency** object with a similar state.

B.2 Interfaces

B.2.1 Currency Interface

FbcCurrency::Currency createCurrencyValue();

CreateCurrencyValue will return an **FbcCurrency::Currency** object that has a state similar to the Currency object's state. This newly created **FbcCurrency::Currency** object can then be passed as a parameter into the **FbcCurrency** operations.

B.2.2 CurrencyFactory Interface

Currency createCurrencyFromValue(in FbcCurrency::Currency currency-Value)
raises(CBO::CBOException);

The **createCurrencyFromValue** method will create a **CboCurrency** currency object using the **FbcCurrency::Currency** object passed in. This newly created currency will contain the same state as the passed-in currency. This would be useful, for example, if an **FbcCurrency** operation returns a value which the user wants to convert to a CBO currency type.

B.2.3 DMoney Interface

FbcCurrency::Money createMoneyValue();

CreateMoneyValue will return an **FbcCurrency::Money** object that has state similar to the **DMoney** object's state. This newly created **FbcCurrency::Money** object can then be passed as a parameter into the **FbcCurrency** operations.

B.2.4 DmoneyFactory Interface

DMoney createDMoneyFromValue(in FbcCurrency::Money moneyValue)
raises(CBO::CBOException);

The **createDMoneyFromValue** method will create a **DMoney** object using the **FbcCurrency::Money** object passed in. This newly created money will contain the same state as the passed-in money. This would be useful, for example, if an **FbcCurrency** operation returns a value which the user wants to convert to a **DMoney** type.

B.2.5 DExchangeRateDateBased Interface

FbcCurrency::ExchangeRate createExchangeRateValue();

CreateExchangeRateValue will return an **FbcCurrency::ExchangeRate** object that has state similar to the state of the **DExchangeRateDateBased** object except date information. This newly created **FbcCurrency::ExchangeRate** object can then be passed as a parameter into the FbcCurrency operations.

B.2.6 ExchangeRateFactory Interface

DMoney createDExchangeRateDateBasedFromValue(in FbcCurrency::ExchangeRate exchangeRateValue)
raises(CBO::CBOException);

The **createDExchangeRateDateBasedFromValue** method will create a **DExchangeRateDateBased** object using the **FbcCurrency::ExchangeRate** object passed in. This newly created exchange rate will contain the same state as the passed-in exchange rate, except for date. The user would need to set the date information using the accessor methods on the new object. This would be useful, for example, if an **FbcCurrency** operation returns a value which the user wants to convert to a **DExchangeRateDateBased** type.

B.3 Compliance Level 2 CboCurrency IDL

```
#ifndef _CboCurrency_idl_
#define _CboCurrency_idl_

#include <CBO.idl>
#include <CORBA.idl>
#include <CosQueryCollection.idl>
#include <FbcCurrency.idl>

module CboCurrency {
   // exceptions
   // exception CBOException; is picked up from CBO.IDL

   //  Interface Definitions
   interface Currency;
   value DMoney;
   value DExchangeRateDateBased;

   interface Currency
   {
      FbcCurrency::Currency createCurrencyValue();

      void   initializeCurrency(
                      in wstring name,
                      in wstring mnemonic,
                      in wstring numericCode,
                      in wstring symbol,
                      in wstring fractionSymbol,
                      in long scaleFactor,
                      in short ratioOfFractionToWhole,
                      in wstring description,
                      in CBO::DTime introductionDate,
                      in CBO::DTime expirationDate,
                      in boolean isISO,
                      in wstring ISOVersion,
                      in boolean isExternalOutputShowingFractions,
                      in boolean isinternalOutputShowingFractions)
           raises(CBO::CBOException);

      long  getScaleFactor() raises(CBO::CBOException);
      void   setScaleFactor(in long scaleFactor) raises(CBO::CBOException);

      boolean isExternalOutputShowingFractions() raises(CBO::CBOException);
      void   setExternalOutputShowingFractions(in boolean flag)
```

```
                raises(CBO::CBOException);

        boolean isInternalOutputShowingFractions() raises(CBO::CBOException);
        void   setInternalOutputShowingFractions(in boolean flag)
                raises(CBO::CBOException);

boolean equals(in Object anotherCurrency) raises(CBO::CBOException);

        wstring getMnemonic() raises (CBO::CBOException);
        void setMnemonic(in wstring mne) raises(CBO::CBOException);

        short getNumericCode() raises (CBO::CBOException);
        void setNumericCode(in short numericCode) raises(CBO::CBOException);

        wstring getName() raises (CBO::CBOException);
        void setName(in wstring name) raises (CBO::CBOException);

        wstring getFractionName() raises (CBO::CBOException);
        void setFractionName(in wstring name) raises (CBO::CBOException);

        wstring getSymbol() raises (CBO::CBOException);
        void setSymbol(in wstring symbol) raises (CBO::CBOException);

        wstring getFractionSymbol() raises (CBO::CBOException);
        void setFractionSymbol(in wstring symbol) raises (CBO::CBOException);

        short getRatioOfFractionToWhole() raises (CBO::CBOException);
        void setRatioOfFractionToWhole(in short ratio)
                            raises (CBO::CBOException);

        CBO::DTime getIntroductionDate() raises (CBO::CBOException);
        void setIntroductionDate(in CBO::DTime date) raises (CBO::CBOException);

        CBO::DTime getExpirationDate() raises (CBO::CBOException);
        void setExpirationDate(in CBO::DTime date) raises (CBO::CBOException);

        wstring getDescription() raises (CBO::CBOException);
        void setDescription(in wstring description) raises (CBO::CBOException);

        boolean isISOCurrency() raises (CBO::CBOException);
        wstring getISOVersion() raises (CBO::CBOException);

        CosQueryCollection::Iterator getLocaleIterator() raises (CBO::CBOException);
        void addLocale(in wstring locale)
           raises (CBO::CBOException); // UNKNOWN_LOCALE

        void removeLocale(in wstring locale)
            raises (CBO::CBOException); // UNKNOWN_LOCALE

    };

    value DMoney
    {
        FbcCurrency::Money createMoneyValue();
        // Initialization and Setting/Getting Data
        //
        void   initializeDMoney(in Currency theCurrency,
                          in CBO::DDecimal theValue)
              raises(CBO::CBOException);

        Currency   getCurrency() raises(CBO::CBOException);
        void setCurrency(in Currency newCurrency) raises (CBO::CBOException);

        long   getScaleFactor() raises (CBO::CBOException);

    wstring getCurrencyMnemonic() raises (CBO::CBOException);

        CBO::DDecimal getValue() raises (CBO::CBOException);
        void setValue(in CBO::DDecimal amount) raises (CBO::CBOException);

    // DMoney Arithmetic
        //
        DMoney add(in DMoney anotherMoney) raises (CBO::CBOException);
        DMoney subtract(in DMoney anotherMoney) raises (CBO::CBOException);
        DMoney multiply(in CBO::DDecimal multiplier) raises (CBO::CBOException);
        DMoney divide(in CBO::DDecimal denominator) raises (CBO::CBOException);

        // DMoney Logical Operations
        //
        boolean equals(in Object anotherMoney) raises (CBO::CBOException);
        boolean greater(in DMoney anotherMoney) raises (CBO::CBOException);
        boolean greaterEqual(in DMoney anotherMoney) raises (CBO::CBOException);
        boolean greaterThanZero() raises (CBO::CBOException);
        boolean less(in DMoney anotherMoney) raises (CBO::CBOException);
        boolean lessEqual(in DMoney anotherMoney) raises (CBO::CBOException);
        boolean lessThanZero() raises(CBO::CBOException);
        boolean isZero() raises(CBO::CBOException);
        boolean isOfSameCurrency(in DMoney anotherMoney) raises(CBO::CBOException);

        // DMoney Rounding and Truncation etc.
        //
        DMoney createRounded(in long places) raises(CBO::CBOException);
        DMoney createTruncated(in long places) raises(CBO::CBOException);
        DMoney createCeiling() raises(CBO::CBOException);
        DMoney createFloor() raises(CBO::CBOException);
        DMoney createAbs() raises(CBO::CBOException);
        DMoney createChangeSign() raises(CBO::CBOException);


    };

    value DExchangeRateDateBased
    {
        FbcCurrency::ExchangeRate createExchangeRateValue();

        void initializeDExchangeRateDateBased( in wstring rateTypeID,
                          in Currency fromCurrency,
                          in Currency toCurrency,
                          in CBO::DTime startDate,
                  in CBO::DTime endDate,
                  in CBO::DDecimal conversionFactor)
             raises(CBO::CBOException);

    Currency   getSourceCurrency() raises (CBO::CBOException);
    void   setSourceCurrency(in Currency newSourceCurrency)
             raises(CBO::CBOException);

    Currency   getTargetCurrency() raises (CBO::CBOException);
    void   setTargetCurrency(in Currency newTargetCurrency)
             raises(CBO::CBOException);

    CBO::DTime getStartDate() raises (CBO::CBOException);
    void setStartDate(in CBO::DTime startDate)
             raises(CBO::CBOException);

    CBO::DTime getEndDate() raises (CBO::CBOException);
    void setEndDate(in CBO::DTime endDate)
             raises(CBO::CBOException);

    wstring getSourceCurrencyMnemonic() raises (CBO::CBOException);
    void setSourceCurrencyMnemonic(in wstring currencyMnemonic)
             raises(CBO::CBOException);

    wstring getTargetCurrencyMnemonic () raises(CBO::CBOException);
    void setTargetCurrencyMnemonic(in wstring currencyMnemonic)
             raises(CBO::CBOException);

    CBO::DDecimal getConversionFactor() raises(CBO::CBOException);
    void setConversionFactor(in CBO::DDecimal conversionFactor)
             raises(CBO::CBOException);

    wstring getType() raises(CBO::CBOException);
    void setType(in wstring exchangeRateType)
                      raises(CBO::CBOException);
                      boolean equals(in Object another ExchangeRate)
                             raises(CBO::CBOException);

    DMoney exchange(in DMoney sourceMoney) raises(CBO::CBOException);

};

interface CurrencyFactory
{
    Currency createCurrencyFromValue(in FbcCurrency::Currency currencyValue)
               raises(CBO::CBOException);

    Currency createCurrency(in wstring name,
                    in wstring mnemonic,
                    in wstring numericCode,
                    in wstring symbol,
                    in wstring fractionSymbol,
                    in long scaleFactor,
                    in short ratioOfFractionToWhole,
                    in wstring description,
                        in CBO::DTime introductionDate,
                        in CBO::DTime expirationDate,
                        in boolean isISO,
                        in wstring ISOVersion,
                        in boolean isExternalOutputShowingFractions,
                        in boolean isInternalOutputShowingFractions)
                   raises(CBO::CBOException);


    };

    interface MoneyFactory
    {
        DMoney createDMoneyFromValue(in FbcCurrency::Money moneyValue)
                raises(CBO::CBOException);

        DMoney createDMoney(in Currency theCurrency,
                    in CBO::DDecimal theValue) raises(CBO::CBOException);

    };

    interface  DExchangeRateDateBasedFactory
    {

        DExchangeRateDateBased createDExchangeRateDateBasedFromValue(
                in FbcCurrency::ExchangeRate exchangeRateValue)
                raises(CBO::CBOException);

        DExchangeRateDateBased createDExchangeRateDateBased(
                in wstring rateTypeID,
                in Currency fromCurrency,
                in Currency toCurrency,
                in CBO::DTime startDate,
                in CBO::DTime endDate,
                in CBO::DDecimal conversionFactor)
                raises (CBO::CBOException);


     };

}; // end of CboCurrency Module
#endif // _CboCurrency_idl
```

Index

Index