General Ledger Specification

February 2001
Version 1.0

Copyright 1999, Blueprint Technologies, Inc.
Copyright 1999, Economica AS.
Copyright 1999, Real Objects Ltd.
Copyright 1999, SINTEF AS.
Copyright 1999, Stanford Software International Ltd.
Copyright 1999, The Software Box Ltd.

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

PATENT

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

NOTICE

The information contained in this document is subject to change without notice. The material in this document details an Object Management Group specification in accordance with the license and notices set forth on this page. This document does not represent a commitment to implement any portion of this specification in any company's products.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIEDWARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR PARTICULAR PURPOSE OR USE. In no event shall The Object Management Group or any of the companies listed above be liable for errors contained herein or for indirect, incidental, special, consequential, reliance or cover damages, including loss of profits, revenue, data or use, incurred by any user or any third party. The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Right in Technical Data and Computer Software Clause at DFARS 252.227.7013 OMG® and Object Management are registered trademarks of the Object Management Group, Inc. Object Request Broker, OMG IDL, ORB, CORBA, CORBAfacilities, CORBAservices, and COSS are trademarks of the Object Management Group, Inc. X/Open is a trademark of X/Open Company Ltd.

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form at *http://www.omg.org/library/issuerpt.htm*.

Contents

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

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 Language Mappings*, 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* is a collection of services that many applications may share, but which are not as fundamental as the Object Services. For instance, a system management or electronic mail facility could be classified as a common facility.
- *CORBA Manufacturing*: Contains specifications that relate to the manufacturing industry. This group of specifications defines standardized object-oriented interfaces between related services and functions.
- *CORBA Med*: Comprised of specifications that relate to the healthcare industry and represents vendors, healthcare providers, payers, and end users.
- *CORBA Finance*: Targets a vitally important vertical market: financial services and accounting. These important application areas are present in virtually all organizations: including all forms of monetary transactions, payroll, billing, and so forth.
- *CORBA Telecoms*: Comprised of specifications that relate to the OMG-compliant interfaces for telecommunication systems.

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
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
pubs@omg.org
http://www.omg.org

Acknowledgments

The following companies submitted and/or supported parts of this specification:

- Blueprint Technologies, Inc.
- Economica AS
- Real Objects Ltd.
- SINTEF AS
- Stanford Software International Ltd.
- The Software Box Ltd.

Overview 1

Contents

This chapter contains the following topics.

1.1 GL Facility - Description

The OMG General Ledger Facility defines the interfaces and their semantics that are required to enable interoperability between General Ledger systems and accounting applications, as well as other distributed objects and applications for accounting purposes.

The business accounting function (of which, General Ledger is the common core) is a statutory requirement for all commercial organizations and individual proprietorship. The vast majority of General Ledger systems are proprietary, non-standard and non-interoperable, even though the underlying accounting concepts have been stable for over 500 years. Applications such as Payroll systems and Report Writers need to interoperate with General Ledger systems, however; this is currently a tedious, difficult, and error prone task due to the general lack of technology standardization. Additionally, many other accounting applications including Accounts Payable, Accounts Receivable, Inventory, Sales and Purchase Order Processing, and Invoicing also need to interoperate with General Ledger systems. Standard interfaces to General Ledger would allow the user to mix and match different vendors’ implementations of accounting applications, and enable interoperability with other kinds of applications.

Figure 1-1 Ludwig von Mises Quotation

“Monetary calculation is the guiding star of action under the social system of division of labor. It is the compass of the man embarking upon production ... [It] is the main vehicle of planning and acting in the social setting of a society of free enterprise directed and controlled by the market and its prices ... Our civilization is inseparably linked with our methods of economic calculation. It would perish if we were to abandon this most precious intellectual tool of acting. Goethe was right in calling book-keeping by double entry ‘one of the finest inventions of the human mind’.” - Ludwig von Mises, Human Action: A Treatise on Economics, Regnery, 1963.

1.2 GL Facility - Approach

The General Ledger (GL) Facility specifies interfaces that encapsulate distributed object frameworks implementing accounting General Ledgers; these GLs are conformant with International Accounting Standards for double entry book-keeping. The GL interfaces comprise a framework (in the object-oriented sense) that supports the implementation of accounting client applications (for example, Accounts Payable, Accounts Receivable, Payroll, and so forth).

The architectural intention is to facilitate the convenient implementation of interoperable accounting applications, referred to as “clients” in this specification. The overall intention is to provide as complete a set of GL services as possible in order to support the implementation of accounting clients that need to interoperate with one or more GL Facility implementations.

All user interfaces are the responsibility of the clients whereas GL Facility implementations are responsible for back-end operations. The GL Facility supports various GL characteristics and operations such as persistence, multi-currency, and other requirements specified by the Object Management Group's General Ledger Facility Request for Proposal, as recommended by the OMG Financial Domain Task Force (FDTF), the OMG Accounting Working Group, the European Union’s Esprit COMPASS project, and many others.

1.3 GL Facility - Interface Summary

The General Ledger Facility defines interfaces (using OMG IDL | ISO DIS 14750) to support the capabilities as outlined previously. Ta ble1-1 gives a high level description of the General Ledger (GL) Facility interfaces. Subsequent sections describe the GL Facility interfaces in more detail.

Table 1-1 General Ledger Interfaces

Interface	Purpose	Primary GL Client(s)
GL Arbitrator	GL client session establishment	all GL clients
GL Profile	Discriminatory access to GL services	GL client session
GL BookKeeping	GL Transaction entry	GL data entry clients
GL Retrieval	GL information Retrieval operations	GL reporting clients
GL LedgerLifecycle	GL Lifecycle operations	GL administration clients
GL Integrity	GL Integrity checks	GL administration clients
GL FacilityLifecycle	GL Facility Lifecycle operations	GL administration clients

1.4 GL Facility - Architecture

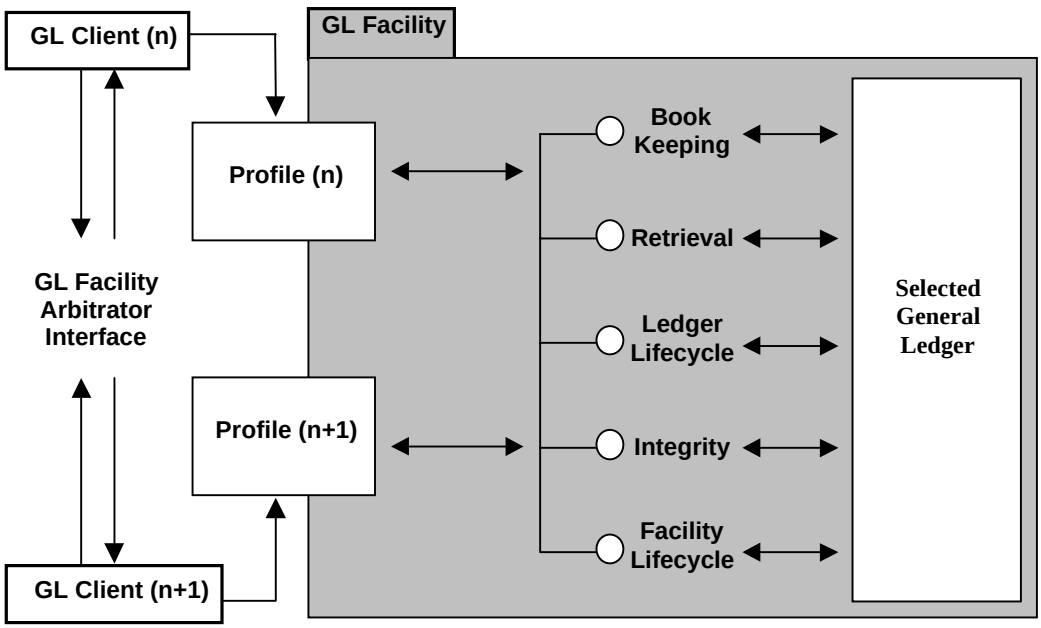


Figure 1-2 Different Interfaces that Comprise the GL Facility

Figure 1-2 illustrates the different interfaces that comprise the GL Facility, as documented previously in Table 1-1. The intention of the diagram is to show that one or more GL client(s) makes a request for a new GL client session from the GL Facility's Arbitrator interface. If the invocation was successful, the GL

Arbitrator::open_session() operation returns a GL Profile interface that provides information about the current session, as well as making provision for controlled access to the various interfaces and operations supported by the GL Facility.

Modules and Interfaces 2

Contents

This chapter contains the following topics.

2.1 Module - CORBA::FdGeneralLedger

The **FdGeneralLedger** module specifies the interfaces of the GL Facility, as well as the structs, exceptions, and typedefs used by those interfaces. The interfaces are defined for different types of client applications and users, so that a GL client does not have to depend upon interfaces it doesn’t use.

2.1.1 GL Facility - Included OMG/ISO IDL Files

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

The GL Facility uses the Currency type as defined by the OMG Currency Facility specification, and uses the DTime type defined by the OMG Common Business Objects specification. See Appendix A for more information.

2.1.2 GL Facility - Module FdGeneralLedger

module FdGeneralLedger {

The module statement establishes the syntactic scope for the GL Facility definitions. The module name follows the OMG Financial Domain Task Force naming standards by prefixing the module name with "**Fd**."

2.1.3 GL Facility - Environment Contract

The following bulleted items are the key assumptions provisioned for the environmental objects containing and managing the GL Facility.

- The GL Facility assumes that GL client authentication for the security policy domain has occurred prior to access to GL Facility interfaces. See Appendix A for more information.
- The GL Facility assumes that access controls will be applied according to system domain policies prior to and during GL client sessions. For example, the passing of clear-text parameters in operation invocations will be protected from unauthorized access or disclosure.
- The only interface provided to GL clients prior to GL session establishment is the GL Arbitrator interface. The environment shall not disclose other GL interfaces to GL clients. That is the responsibility of the GL Arbitrator interface. For example, only the GL Arbitrator interface shall be advertised in the Trader Service and Name Service. Other GL interfaces are provided by the GL Profile interface, subsequent to GL client session establishment.

2.1.4 GL Facility - General Invariants

This section details key assumptions regarding the responsibilities of GL Facility implementations, as well as detailing various invariants for General Ledgers, GL Accounts, GL Transactions, and GL Entries.

The GL Facility maintains state for each client session. For example, each GL client session concerns only one known General Ledger and that General Ledger's established Chart of Accounts.

There is a one-to-one mapping between each General Ledger and each Chart of Accounts in each GL Facility instance. This “single set of books” constraint is conformant with International Accounting Standards. However, a GL Facility is not responsible for enforcing this constraint in federation with other GL Facility installations.

Operations performed during each GL client session are constrained by session-specific GL policies. See Section 2.9, “GL Facility - FacilityLifecycle Interface,” on page 2-48 for administrative operations.

Implementations make available at least one Chart of Accounts to GL clients so that an initial GL client session may be established in situations where no General Ledgers exist. This chart may either be empty or a default chart. Implementations with zero General Ledgers are prohibited.

2.1.5 GL Facility - Ledger Invariants

The specified names of General Ledgers to be constrained and managed by the GL Facility must be unique. The duplication of General Ledger names is prohibited. The naming of specific General Ledgers is usually carried out by privileged GL Facility administrators.

The decision to remove a specific General Ledger, GL Account, GL Transaction, or GL Entry from the GL Facility is set by organizational and environment specific policies.

The currently selected General Ledger’s reporting currency is determined by organizational policy. A reporting currency is the nominated currency used by a reporting enterprise and is used in the presentation of general purpose financial statements. The reporting enterprise is an enterprise or other organizational entity, for which there are users who rely on general purpose financial statements as their major source of information about the enterprise. A single General Ledger constrained by a GL Facility implementation may support only a single reporting currency.

Individual GL Transactions and GL Entries must be “date stamped” in order to comply with Generally Accepted Accounting Principles (GAAP). However, these date stamps can have very different meanings and therefore impact. For example, GL Transactions and GL Entries have an “actual date” that represents a “physical” or “actual” calendar date. However, this date is not necessarily the same date as the “logical” accounting date or “entry date” also associated with a GL Entry. It is assumed that GL Facility implementations will support the GL Retrieval (and other) GL interface operations based on either or both of the date semantics associated with GL Transactions and GL Entries.

2.1.6 GL Facility - Account Invariants

GL Account references constrained by a specific General Ledger’s Chart of Accounts must be unique. The duplication of GL Account references is prohibited.

The decision to remove a specific GL Account from a specific General Ledger constrained and managed by the GL Facility is set by organizational and environment specific policies. The specific GL Account must exist and cannot be in use. GL Accounts

that are in use cannot be removed. A GL Account is in use when there are associated GL Transactions and/or GL Entries in the specified General Ledger or the balance of the account is non-zero.

GL Accounts must be created with an account balance equal to zero. The monetary amount represented by the balance of a single GL Account that has no associated GL Transactions and/or GL Entries can only be equal to zero. A newly created GL Account must be created first, prior to the posting of any GL Transactions and GL Entries to be associated with a specified GL Account. The creation of new GL Accounts with a non-zero balance is prohibited.

GL Account references may not be modified in any way. The only way to modify a GL Account reference is by removing it before it is in use. A specified GL Account cannot be modified if it is in use. The modification of GL Account references is prohibited.

2.1.7 GL Facility - Transaction Invariants

GL Transaction references must be unique. The duplication of GL Transaction references is prohibited. GL Transaction references may not be removed or modified in any way. The modification and removal of GL Transaction references is prohibited.

The sum of all GL Entry debits and credits constrained by any GL Transaction must be equal (i.e., “balanced”). GL Transactions that are unbalanced are prohibited. If the debit and credit accounts of the GL Entries constrained by a single GL Transaction are the same, implementations will ensure that the debit takes place before the credit in the audit trail.

GL Transaction references may be ignored by implementations that automatically allocate these references at the server.

The GL Transaction **BookKeeping::post_transaction_list()** operation return values must match the corresponding **TransactionIdList** for implementations, which allocate GL Transaction references at the client.

2.1.8 GL Facility - Entry Invariants

A single GL Entry constrained by a single GL Transaction and posted to a specific General Ledger constrained by the GL Facility has an identifier that must be unique. This identifier represents the “audit trail” number of each GL entry.

GL Entries may not be removed or modified in any way. A GL Entry that has been posted to any General Ledger constrained and managed by an instance of the GL Facility cannot be removed or modified. The modification and/or removal of GL Entries is prohibited.

A single GL Entry constrained by a single GL Transaction and posted to a specific General Ledger has an identifier that specifies whether the GL Entry was a debit or credit. A single GL Entry can only be a debit or credit.

A single GL Entry constrained by a single GL Transaction and posted to a specific General Ledger constrained and managed by an instance of the GL Facility has an identifier that specifies the precise nature of that GL Entry. Implementations claiming conformance with this specification provide each GL Entry a type with a unique GL Entry type code, which semantically differentiates between the various types of GL related financial accounting transactions (for example, “Journal Debit” or “Journal Credit.”)

A single GL Entry constrained by a single GL Transaction and posted to a specific General Ledger constrained and managed by the GL Facility will record details of the amount (debited or credited) in a foreign currency, as well as in the nominated reporting currency for a specific General Ledger.

2.1.9 GL Facility - Forward Declarations

Forward declarations are included for all of the interfaces defined in the GL Facility.

```
interface Arbitrator;          // establish GL client session
interface Profile;             // GL Facility metadata
interface BookKeeping;         // GL data entry
interface Retrieval;           // GL data acquisition
interface LedgerLifecycle;     // GL lifecycle management
interface Integrity;           // GL data integrity checks
interface FacilityLifecycle;   // GL Facility lifecycle management
```

2.1.10 GL Facility - Basic Data Type Naming Conventions

The FdGeneralLedger module defines several data types for GL accounting information. Among these are a number of **sequence** types, which follow the naming convention **<T>List** where **<T>** is the type of the **sequence** elements. All string types used in the module are wstrings.

2.1.11 GL Facility - Basic Data Type Definitions

```
typedef FbcCurrency::Money Money;       // imported from the OMG Currency
                                        // specification, module FbcCurrency
typedef sequence<wstring> wstringList;  // sequence of type wstring
typedef sequence<boolean> booleanList;  // sequence of type boolean
typedef sequence <octet> octetList      // used for authorization code

typedef wstring CurrencyMnemonic;       // a reference to a single currency

typedef wstring TransactionId;          // a single GL Transaction reference
typedef sequence<TransactionId> TransactionIdList;

typedef wstring EntryId;                // audit trail number for a GL Entry
typedef sequence<EntryId> EntryIdList;

typedef wstring AccountId;              // a single GL Account reference
typedef sequence<AccountId> AccountIdList;
```

```
typedef wstring EntryType;              // a single GL Entry type
typedef sequence<EntryType> EntryTypeInfoList;

typedef wstring PeriodId;               // a reference to a user defined period
```

2.1.12 GL Facility - Basic Data Type Information

```
typedef unsigned short ChartKind;
    const ChartKind EMPTY_CHART    = 1;
    const ChartKind DEFAULT_CHART  = 2;
    const ChartKind EXISTING_CHART = 3;
```

An instance of the GL Facility constrains and manages one or more General Ledgers, each of which requires a single Chart of Accounts (schema) to be established before proper operation can be assured. The data type **ChartKind** specifies the different kinds of Chart of Accounts schemata for the initialization of a new General Ledger, which is to be constrained and managed by an instance of the GL Facility. Please refer to the Section 2.9, “GL Facility - FacilityLifecycle Interface,” on page 2-48 interface operation **create_ledger_chart_of_accounts()** for specific information.

Each General Ledger has an identifying name and a single Chart of GL Accounts schema. A new General Ledger may be initialized without first specifying the Chart of Accounts. That is, the new General Ledger is created but has zero GL Accounts defined. This schema type is represented by the **EMPTY_CHART** member of the **ChartKind** data type.

A single Chart of GL Accounts schema (associated with a single General Ledger) may be required to be conformant with specific accounting practices and/or regulations in a particular locale or country, for example. This schema type is represented by the **DEFAULT_CHART** member of the **ChartKind** data type. Governments and other regulatory bodies of the vast majority of countries in the world have specific minimum requirements regarding a General Ledger’s Chart of Accounts schema.

A new General Ledger may be based on an existing Chart of Accounts in another General Ledger managed by the GL Facility. This schema type is represented by the **EXISTING_CHART** member of the **ChartKind** data type. The new Chart of Accounts may be revised subsequently by using the administrative operations provided by the GL FacilityLifecycle interface.

```
enum DebitOrCredit { DEBIT, CREDIT };
```

The data type **DebitOrCredit** specifies whether a single GL Entry is a debit or a credit. A single GL Entry can be either a debit or a credit.

```
Struct Date { CBO::DTime setting;
              boolean is_set; };
```

The data type **Date** is used for all date operations. However, the **CBO::DTime** does not allow for "not set" date values, which are frequently needed for many date oriented operations. The boolean **is_set** is added to provide a workaround for many frequently encountered situations that require a blank date, such as the retrieval of information from legacy systems.

```
struct DateRange { Date start_date; Date end_date; };
```

The data type **DateRange** specifies date ranges for the purpose of supporting general purpose retrieval operations on GL accounting information such as GL Transactions and GL Entries, for example. The **DateRange** data type is inclusive of both a **start_date** and **end_date**.

2.1.13 GL Facility - Account Information

The GL Facility is responsible for one or more General Ledgers. A single, specific General Ledger must constrain and manage a single collection of GL Accounts, also known as a "Chart of Accounts."

Subsequently, GL Transactions and their associated GL Entries may be further associated with one or more GL Accounts (specified in the currently selected General Ledger's Chart of Accounts schema) by "posting" (i.e., recording) GL Transactions concerning at least two (or more) GL Accounts contained in the General Ledger's Chart of Accounts schema. Please see the various GL **LedgerLifecycle** interface operations for specific information about General Ledger administrative operations as well as the GL **Profile::BookKeeping** interface operations for more information about posting GL Transactions.

```
struct AccountInfo {    AccountId acc_id;
    wstring acc_description;
};
typedef sequence<AccountInfo> AccountInfoList;
```

The data type **AccountInfo** comprises a single GL Account reference that is unique to a specific General Ledger, as well as a user defined GL Account name, which may be used to describe the nature or purpose for which a particular GL Account will be used. For example, salaries, bank current account. The data type **AccountInfoList** is a collection type for the data type **AccountInfo**.

```
struct Account {
        AccountInfo acc_info;
        boolean is_control_account;
        Money balance;
};
typedef sequence<Account> AccountList;
```

The data type **Account** specifies summary information describing a single GL Account. The data type **AccountList** is a collection type for GL Accounts.

The **acc_info** member of the data type **AccountInfo** specifies a unique GL Account reference and name describing the name or purpose of this particular GL Account (for example, salaries or bank current account).

If a GL Account has been specified as a “Control Account,” the member **is_control_account** will always return TRUE. A GL Control Account accumulates a monetary value of related subsidiary GL Accounts over time. Some examples of GL Control Account designations could be Debtor's Control Account, Creditor's Control Account, Inventory, Buildings, and Equipment.

The GL Account member balance is a state value representing an accumulated monetary value as recorded in the currently selected General Ledger, to date. The GL Account balance represents a monetary amount in the currently selected General Ledger’s reporting currency.

2.1.14 GL Facility - Transaction Information

```
struct  TransactionInfo {
      TransactionId trans_id;
          wstring voucher_ref;
          Date voucher_date;
          Date act_trans_date;
          PeriodId period_id;
};
typedef sequence <TransactionInfo> TransactionInfoList;
```

The data type **TransactionInfo** specifies summary information about a single, specific GL Transaction. The data type **TransactionInfoList** is a collection type for multiple **TransactionInfo** data types

The member **trans_id** specifies a unique GL Transaction reference for a single, specific GL Transaction.

The member **voucher_ref** refers to any external (e.g., physical) documentation that may be related to (or associated with) a single GL Transaction such as a check, purchase order, or invoice number. Many National statutory regulations require that each GL Transaction in the accounting books must correspond to a “real” business transaction (usually represented by a paper voucher) and that it must also be easy to trace a specific GL Entry in the books to a corresponding voucher. This information may also be required for internal auditing purposes.

The member **voucher_date** refers to the date of the “real” business transaction that caused the GL Transaction to be created and posted to the General Ledger. For example, a date on some external documentation (e.g., voucher) that a GL Transaction is to be (or was) associated with. Please see the **TransactionInfo** member **voucher_ref** as mentioned previously.

The member **act_trans_date** refers to the “physical” or actual calendar date that the GL Transaction was (or is to be) posted to the currently selected General Ledger, and may be used for general purpose GL Transaction retrieval.

The member **period_id** refers to an implementation-defined accounting period for the logical posting of this GL Transaction. The data type **PeriodId** is of type **wstring**.

Generally, financial accounting is done within “accounting periods.” An accounting period is defined as a period of time (of arbitrary length), whose only constraint is that a given accounting period is shorter than a full financial year. This applies not only to General Ledger accounting, but also to other often related accounting functions such as Accounts Receivable and Accounts Payable. The set up of accounting periods is typically performed only once a year and is the subject of organizational policy. This specification makes no prescriptive statements about the rules that may be applied when using accounting periods.

As of this writing, both Period Management as well as Closing and Reconciliation Facilities are being considered as potential candidates for future OMG FDTF RFPs.

2.1.15 GL Facility - Entry Information

Each GL Transaction is composed of at least two (or more) GL Entries. Every GL Entry in the GL Facility corresponds to either a debit or a credit. Each GL Entry contains the details of an amount (debited or credited) in a foreign currency, as well as that same amount in the nominated reporting currency of a specific General Ledger. In combination with the historical exchange rate information provided by an instance of the OMG Currency Facility, this provides sufficient information to generate multi-currency general purpose financial statements, should this be required.

```
struct Entry {
      TransactionId trans_id;
          EntryId entry_id;
          Date entry_date;
          EntryType entry_type;
          AccountId acc_id;
          Money orig_amount
          Money amount;
          DebitOrCredit dr_cr;
          wstring description;
          wstring voucher_ref;
};
typedef sequence<Entry> EntryList;
```

The data type **EntryList** specifies a collection type for multiple GL Entries. The data type **Entry** specifies information about a single GL Entry.

The member **trans_id** specifies the parent GL Transaction that a single GL Entry is associated with and is synonymous with the **TransactionId** data type of the **TransactionInfo** data type mentioned previously.

The member **entry_id** specifies the “audit trail” number of a GL Entry, while the members **entry_date** and data type **entry_type** (respectively) specify the “logical” date that a GL Entry was (or is to be) posted and the GL Entry type identifying the type of a single GL Entry.

The member **acc_id** specifies the GL Account associated with a single GL Entry.

The member **orig_amount** specifies the original amount of money of a foreign currency business transaction. A "foreign currency" is defined as a currency other than the reporting currency of an enterprise. See Section 2.3.8, "GL Profile Interface Operation ::get_ledger_currency()," on page 2-22 for more information.

The member Amount specifies the monetary amount of a GL Entry in a General Ledger's reporting currency, while the member **dr_cr** indicates whether the GL Account involved was (or is to be) debited or credited.

The GL Entry members description and **voucher_ref** hold user defined annotation that describes the GL Entry, and a voucher reference to any relevant documentation associated with the GL Entry.

```
struct EntryTypeInfo {
        EntryType       type;
        wstring         description;
};
    typedef sequence<EntryTypeInfo> EntryTypeInfoList;
```

The data type **EntryTypeInfo** refers to a single, specific GL Entry type and an associated high level description such as "Journal Debit" or "Journal Credit," for example. The data type **EntryTypeInfoList** is a collection type of **EntryTypeInfo**.

The member type refers to a single GL Entry type, which represents the type of financial accounting transaction associated with a single, specific GL Entry.

The member description refers to a high level, human-readable description such as "Journal Credit" or "Journal Debit."

2.1.16 GL Facility - Transaction Summary Information

```
struct Transaction {
     TransactionInfo trans_info;
        EntryList entries;
    };
     typedef sequence<Transaction> TransactionList;
```

The data type Transaction specifies a collection type for a single GL Transaction and its associated GL Entries. The data type **TransactionList** specifies a collection type for multiple GL Transactions and their associated GL Entries.

2.1.17 GL Facility - Operation Exception Conditions

This section gives detailed information about the various operation exception conditions that may be raised by the various GL Facility interface operations. The exception declaration comes first, immediately followed by information specific to the exception. Many of these exceptions are used both individually and in combination in this specification. These exceptions can only be raised by the various GL Facility operations described in this specification and are distinct from CORBA "system exceptions."

```
exception BadDate {
        wstring error;
        Date bad_value; };
```

A single **Date** data type passed as a parameter in an operation invocation has been deemed to be invalid and has been rejected. A descriptive error message is returned as a **wstring**, along with the value of the bad **Date** that was rejected. The following non-exhaustive list suggests reasons as to why this exception may be raised:

- end date before start date
- invalid date format
- invalid date
- unknown century.

```
exception BadChartKind {
        wstring error;
        ChartKind bad_value; };
```

A single **ChartKind** data type passed as a parameter in an operation invocation has been deemed to be invalid and has been rejected. A descriptive error message is returned as a **wstring**, along with the value of the bad **ChartKind** that was rejected. See “GL FacilityLifecycle Interface Operation ::create_ledger_chart_of_accounts()” on page 2-49 for more information.

```
exception BadTransaction {
        wstring error; };
```

A single GL Transaction passed as a parameter in an operation invocation has been deemed to be invalid and has been rejected. A descriptive error message is returned as a **wstring**. The exception BadTransaction will be raised if either the GL Transaction or its associated GL Entries have illegal or otherwise invalid values or if the GL Entries constrained by the GL Transaction are not balanced. See Section 2.5.2, “GL BookKeeping Interface Operation ::post_transaction(),” on page 2-38 for more information.

```
exception BadTransactionsInList {
      wstring error;
        booleanList is_bad; };
```

One or more GL Transactions in a collection of GL Transactions has a problem. A descriptive error message is returned as a **wstring**, along with the member **is_bad**, which indicates the position in the list of one or more problem GL Transactions. The exception BadTransactionsInList will be raised if any of the GL Transactions or their GL Entries have illegal or otherwise invalid values, or if any of the sets of GL Entries constrained by the GL Transactions are not balanced.

Please note that the **booleanList** returned by this exception must be at least the same length as the list of GL Transactions passed as an input parameter in relevant operation

invocations. See Section 2.5.3, “GL BookKeeping Interface Operation ::post_transaction_list(),” on page 2-40 for more information. If the BadTransactionsInList exception is raised, then none of the GL Transactions in the list may be posted.

```
exception BadEntryType {
        wstring error;
    EntryType bad_value; };
```

A single **EntryType** data type passed in an operation invocation related to GL Entry types has been deemed to be invalid and has been rejected. A descriptive error message is returned as a **wstring**, along with the value of the bad **EntryType** that caused the problem. See Section 2.3.9, “GL Profile Interface Operation ::get_entry_types(),” on page 2-23 and Section 2.6.6, “GL LedgerLifecycle Interface Operation ::set_entry_types(),” on page 2-45 for more information.

```
exception BadEntryTypeInfoList {
    wstring error;
    EntryTypeInfoList bad_values; };
```

One or more of the GL **EntryType** data types passed in an operation invocation related to GL Entries is deemed to be invalid and has been rejected. A descriptive error message is returned as a **wstring**, along with a list of bad **EntryType**s that caused the problem. See Section 2.6.6, “GL LedgerLifecycle Interface Operation ::set_entry_types(),” on page 2-45 for more information.

```
exception BadCurrencyMnemonic {
    wstring error;
       CurrencyMnemonic bad_value; };
```

The exception BadCurrencyMnemonic is raised if the **CurrencyMnemonic** used in the **LedgerLifecycle::set_ledger_currency()** operation is deemed to be invalid.

```
exception BadAccountId {
    wstring error;
    AccountId bad_value; };
```

A single GL Account has an associated GL Account reference that uniquely distinguishes it from other GL Accounts. There is a problem with a GL Account reference passed in an operation invocation which has caused the exception to be raised. A descriptive error message is returned as a **wstring**, along with the value of the **AccountId** that caused the problem.

```
exception BadTransId {
    wstring error;
    TransactionId bad_value; };
```

A single GL Transaction has an associated GL Transaction reference that uniquely distinguishes it from other GL Transactions. There is a problem with a GL Transaction reference passed in an operation invocation, which has caused this exception to be raised. A descriptive error message is returned as a **wstring**, along with the value of the **TransactionId** that caused the problem.

2.1.18 GL Facility - Miscellaneous Operation Exception Conditions

exception CannotRemove { wstring error; };

There is a problem involving an operation invocation that has attempted to remove a GL related entity. A descriptive error message is returned as a **wstring**. This exception is explicitly raised by the **LedgerLifecycle::remove_ledger()** and **remove_account()** operations.

exception PermissionDenied { wstring error; };

The exception PermissionDenied is raised if a GL client session has not been established with the appropriate authorization to invoke an operation. A descriptive error message is returned as a **wstring**. See Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2 for more information.

exception UnknownLedger { wstring error; wstring bad_value; };

There is a problem in accessing the specified General Ledger. A descriptive error message and the name of the specified General Ledger are both returned as **wstring** types. The exception UnknownLedger is explicitly raised if the specified General Ledger is not known or is otherwise unrecognized by the GL Facility.

exception BadIntegritySelection {
wstring error;
wstring bad_value; };

A single GL Integrity check is unknown to the GL Facility or is otherwise invalid and has been rejected. A descriptive error message, as well as the value of the bad integrity selection are both returned as **wstring** types. Please refer to the various GL Facility interface operations in “GL Facility - Integrity Interface” on page 2-46 for more information.

exception BadAccountName {
wstring error;
wstring bad_value; };

There is a problem with a parameter passed in an invocation operation. A descriptive error message is returned as a **wstring**, along with the value that caused the problem, both of which are returned as **wstring** data types. This exception is explicitly raised if the GL Account description is to be duplicated in the currently selected General Ledger.

See Section 2.6.2, “GL LedgerLifecycle Interface Operation ::create_account(),” on page 2-41 and Section 2.6.4, “GL LedgerLifecycle Interface Operation ::modify_account(),” on page 2-43 for more information.

2.2 GL Facility - Arbitrator Interface

The GL Arbitrator is the initial interface used to establish a GL client session. See Section 1.4, “GL Facility - Architecture,” on page 1-3, Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2, and Section 2.1.4, “GL Facility - General Invariants,” on page 2-2 for more information.

A GL client session must be established prior to use of the GL Facility. If the **Arbitrator open_session()** operation completed successfully, a Profile interface is returned which offers access to the various interfaces and operations of the GL Facility. Each GL client session must use a unique instance of the returned Profile interface.

<<Interface>>

Arbitrator

+ get_ledger_names() : wstringList

+ open_session (ledger_name : wstring, authorization_code : octetList) : Profile

2.2.1 GL Arbitrator Interface General Invariants

This section details invariants that are generally applicable to all operations supported by the GL Arbitrator interface.

The GL Facility must have been pre-configured for general purpose usage by a GL Facility Administrator. There must be at least one General Ledger in existence and known to the GL Facility for these interface operations to succeed. The creation of new General Ledgers is facilitated by the GL FacilityLifecycle interface operations. See Section 2.1.3, “GL Facility - Environment Contract,” on page2-2 and Section2.1.4, “ GL Facility - General Invariants,” on page 2-2 for more information.

2.2.2 GL Arbitrator Interface Operation ::get_ledger_names()

wstringList get_ledger_names();

Description

Each GL Facility can manage the General Ledgers of many different types of organizational entities such as for-profit companies, not-for-profit organizations, individual projects, and proprietorship. The GL Arbitrator interface operation **get_ledger_names()** retrieves the complete set of names associated with all General Ledgers constrained and managed by an instance of the GL Facility.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

This operation invocation returns a **wstringList** of General Ledger names that represents the complete set of General Ledgers constrained and managed by the GL Facility.

Exceptions

None

Postconditions

None

2.2.3 GL Arbitrator Interface Operation ::open_session()

```
Profile open_session (
            in wstring ledger_name,
            in octetList authorization_code)
            raises ( UnknownLedger, PermissionDenied );
};
```

Description

This operation establishes a GL client session for subsequent interaction with a specified General Ledger constrained and managed by the GL Facility. If this operation is successful, a GL client session is established with the specified General Ledger. The GL Profile interface operation **get_ledger_names()** returns a **wstringList** of General Ledger names, which represents the complete set of General Ledgers constrained and managed by the GL Facility.

Preconditions

None

Input Parameters

To successfully establish the GL client session, the name of a General Ledger is specified, as well as a valid authorization code provided by an external service, such as the CORBA Security Service. See Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2 and Section 2.1.4, “GL Facility - General Invariants,” on page 2-2.

Output Parameters

None

Return Value

If the operation invocation completed successfully, the **Arbitrator::open_session()** returns a Profile interface that provides GL client session information, as well as providing support for controllable access to the various interfaces and operations of the GL Facility.

Exceptions

The exception UnknownLedger is raised if the specified General Ledger is not known to the GL Facility. The exception PermissionDenied is raised if the GL client does not have permission to access the specified General Ledger.

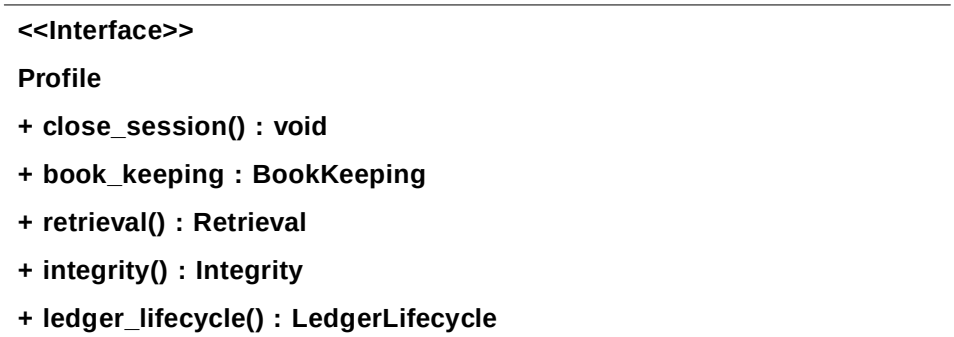
Postconditions

None

2.3 GL Facility - Profile Interface

The GL Arbitrator is the initial interface used to establish a GL client session. A GL client session must be established prior to use of the GL Facility.

If the **Arbitrator::open_session** operation invocation was successful, the Arbitrator returns a **Profile** interface that offers access to the various interfaces and operations of the GL Facility. Each GL client session must use a unique instance of the returned **Profile** interface.



+ facility_lifecycle() : FacilityLifecycle

+ get_ledger_currency() : CurrencyMnemonic

+ get_entry_types() : EntryTypeInfoList

2.3.1 GL Profile Interface General Invariants

This section details invariants that are generally applicable to all operations supported by the GL Profile interface.

A GL client session must have been established with the **Arbitrator::open_session()** operation and each GL client session must use a unique instance of the **Profile** interface. Once the **Profile::close_session()** operation is invoked, all references to other interfaces in the current GL client session immediately become invalid.

The GL Facility must have been pre-configured for general purpose usage and there must also be at least one General Ledger in existence for these operations to succeed. Furthermore, a non empty Chart of Accounts must also exist. Please see Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2 and Section 2.1.4, “GL Facility - General Invariants,” on page 2-2 for more information.

2.3.2 GL Profile Interface Operation ::close_session()

void close_session();

Description

This operation invocation ends the current GL client session, reclaiming session resources and state. Once this operation is invoked and has completed, all references to other interfaces in the current GL client session immediately become invalid. This operation invocation also invalidates any references to the current **Profile** interface so that GL client session resources are properly released.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

None

Exceptions

None

Postconditions

None

2.3.3 GL Profile Interface Operation ::retrieval()

Retrieval retrieval() raises (PermissionDenied);

Description

This operation retrieves a reference to the **Retrieval** interface for use in the current GL client session. If the operation was successful, the various operations of the **Retrieval** interface are made generally available to the GL client for use in the current session.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

This operation invocation returns a **Retrieval** interface for use in the current GL client session. Once the GL client session has been closed, the reference to the **Retrieval** interface is no longer valid.

Exceptions

The exception PermissionDenied is raised if the GL client has not successfully established a valid GL client session with **Arbitrator::open_session()** prior to the call or does not have permission to use this interface.

Postconditions

None

2.3.4 GL Profile Interface Operation ::book_keeping()

BookKeeping book_keeping() raises (PermissionDenied);

Description

This operation retrieves a reference to the **BookKeeping** interface for use in the current GL client session. If the operation was successful, the various operations of the **BookKeeping** interface are made generally available to the GL client for use in the current session.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

This operation invocation returns a **BookKeeping** interface for use in the current GL client session. Once the GL client session has been closed, the reference to the **BookKeeping** interface is no longer valid.

Exceptions

The exception PermissionDenied is raised if the GL client has not successfully established a valid GL client session with **Arbitrator::open_session()** prior to the call or does not have permission to use this interface.

Postconditions

None

2.3.5 GL Profile Interface Operation ::integrity()

Integrity integrity() raises (PermissionDenied);

Description

This operation retrieves a reference to the **Integrity** interface for use in the current GL client session. If the operation was successful, the various operations of the **Integrity** interface are made generally available to the GL client for use in the current session.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

This operation invocation returns an **Integrity** interface for use in the current GL client session. Once the GL client session has been closed, the reference to the **Integrity** interface is no longer valid.

Exceptions

The exception PermissionDenied is raised if the GL client has not successfully established a valid GL client session with **Arbitrator::open_session()** prior to the call or does not have permission to use this interface.

Postconditions

None

2.3.6 GL Profile Interface Operation ::LedgerLifecycle()

LedgerLifecycle ledger_lifecycle() raises (PermissionDenied);

Description

This operation retrieves a reference to the **LedgerLifecycle** interface for use in the current GL client session. If the operation was successful, the various operations of the **LedgerLifecycle** interface are made generally available to the GL client for use in the current session.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

This operation invocation returns a **LedgerLifecycle** interface for use in the current GL client session. Once the GL client session has been closed, the reference to the **LedgerLifecycle** interface is no longer valid.

Exceptions

The exception PermissionDenied is raised if the GL client has not successfully established a valid GL client session with **Arbitrator::open_session()** prior to the call or does not have permission to use this interface.

Postconditions

None

2.3.7 GL Profile Interface Operation ::FacilityLifecycle()

FacilityLifecycle facility_lifecycle() raises (PermissionDenied);

Description

This operation retrieves a reference to the GL **FacilityLifecycle** interface for use in the current client session. If the operation was successful, the various operations of the GL **FacilityLifecycle** interface are made generally available to the GL client for use in the current session.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

This operation invocation returns a **FacilityLifecycle** interface for use in the current GL client session. Once the GL client session has been closed, the reference to the GL **FacilityLifecycle** interface is no longer valid.

Exceptions

The exception PermissionDenied is raised if the GL client has not successfully established a valid GL client session with **Arbitrator::open_session()** prior to the call or does not have permission to use this interface.

Postconditions

None

2.3.8 GL Profile Interface Operation ::get_ledger_currency()

CurrencyMnemonic get_ledger_currency() raises (PermissionDenied);

Description

This operation invocation retrieves the user defined reporting currency of the currently selected General Ledger. If the operation was successful, the GL client has access to the specified reporting currency of the currently selected General Ledger. A reporting currency is the nominated currency used by a reporting enterprise and is used in the presentation of general purpose financial statements. The reporting enterprise is an enterprise or other organizational entity for which there are users who rely on general purpose financial statements as their major source of information about the enterprise.

A “foreign currency” is defined as a currency other than the reporting currency of an enterprise. The exchange rate is defined as the ratio for the exchange of two currencies. A foreign currency transaction should be recorded, on initial recognition in the reporting currency, by applying to the foreign currency amount the exchange rate between the reporting currency and the foreign currency at the date of the transaction. See Section 2.6.5, “GL LedgerLifecycle Interface Operation ::set_ledger_currency(),” on page 2-44 for more information.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

This operation invocation returns a **CurrencyMnemonic** for use in the current GL client session, which specifies the reporting currency for the currently selected General Ledger. The contents of the **CurrencyMnemonic** will be either one of the ISO currency mnemonics defined by the OMG Currency Facility or a specific user-defined mnemonic previously defined using that Facility.

Exceptions

The exception PermissionDenied is raised if the GL client has not successfully established a valid GL client session with **Arbitrator::open_session()** prior to the call or does not have permission to use this interface.

Postconditions

None

2.3.9 GL Profile Interface Operation ::get_entry_types()

EntryTypeInfoList get_entry_types () raises (PermissionDenied);

Description

This operation retrieves a list of GL **Entry** types for the currently selected General Ledger. GL **Entry** types are discrete types assigned to individual GL Entries such as Journal Debits, Journal Credits, and so forth. If the operation was successful, the GL client has access to a list of GL **Entry** types that help describe the purpose of GL Entries in the currently selected General Ledger. See the corresponding operation: Sectio n2.6.6, “GL LedgerLifecycle Interface Operation ::set_entry_types(),” on page 2-45.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

This operation invocation returns an **EntryTypeInfoList**, which specifies all valid GL **Entry** types for the currently selected General Ledger.

Exceptions

The exception PermissionDenied is raised if the GL client has not successfully established a valid GL client session with **Arbitrator::open_session()** prior to the call or does not have permission to use this interface.

Postconditions

None.

2.4 GL Facility - Retrieval Interface

The GL Retrieval interface supports general purpose information retrieval capabilities related to the GL Accounts, Transactions, and Entries in the currently selected General Ledger.

<<Interface>>

Retrieval

+ get_all_account_info() : AccountInfoList

+ get_account (acc_id : AccountId) : Account

+ get_multiple_accounts (account_ids : AccountIdList) : AccountList

+ get_all_accounts() : AccountList

+ get_trans_ids (date_range : DateRange) : TransactionIdList

+ get_trans_info_summary (trans_id : TransactionId) : TransactionInfo

+ get_trans_info_by_refs (trans_ids : TransactionIdList) : TransactionInfoList

+ get_trans_info_by_date (date_range : DateRange) : TransactionInfoList

+ get_entry_count_by_type (date_range : DateRange, entry_type : EntryType) : unsigned long

+ get_entry_count_by_account (date_range : DateRange, acc_id : AccountId) : unsigned long

+ get_transaction() : Transaction

+ get_transactions_by_ids (trans_ids : TransactionList) : TransactionList

+ get_transactions_by_date (date_range : DateRange) : TransactionList

+ get_entries_by_type (date_range : DateRange, entry_type : EntryType) : EntryList

+ get_entries_by_account (date_range : DateRange, acc_id :AccountId) : EntryList

2.4.1 GL Retrieval Interface General Invariants

This section details invariants that are generally applicable to all operations supported by the **GL Retrieval** interface. A GL client session must have been established with the **Arbitrator::open_session()** operation and each GL client session must use a unique instance of the **Profile** interface. The GL Facility must have been pre-configured for general purpose usage and there must also be at least one General Ledger in existence in order for these operations to succeed. Furthermore, a non-empty Chart of Accounts must also exist. See Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2 and Section 2.1.4, “GL Facility - General Invariants,” on page 2-2 for more information.

The **GL Retrieval** interface defines several operations that have a **DateRange** data type passed as an input parameter in the invocation. The **DateRange** data type specifies an inclusive date range with two members, each of which represent a **start_date** and **end_date**, respectively. Implementations claiming conformance with this specification should ensure that all GL Transactions and GL Entries are date stamped; however, in certain situations (such as legacy systems, for example), GL Transactions and GL Entries may have dates that are “not set.” A particular date associated with a single GL Transaction or GL Entry that is “not set,” is considered to be “earlier” than the earliest

date stamp which is “set” (i.e., has a date stamp). Implementations should provide support for the inclusive retrieval of both dated and undated GL Transaction and GL Entry information.

2.4.2 GL Retrieval Interface Operation ::get_all_account_info()

AccountInfoList get_all_account_info()raises (PermissionDenied);

Description

This operation invocation retrieves a sequence of all GL Account references and descriptive names that exist in the currently selected General Ledger. If the operation was successful, the GL client has access to a list of all GL Accounts in the currently selected General Ledger. The data type **AccountInfoList** contains information that describes the name and purpose of all GL Accounts in the currently selected General Ledger.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

This operation invocation returns an **AccountInfoList**, each member of which contains a GL Account reference and descriptive name (i.e., purpose) of an individual GL Account, for all GL Accounts in the currently selected General Ledger.

Exceptions

The exception PermissionDenied is raised if the GL client has not successfully established a valid GL client session with **Arbitrator::open_session()** prior to the call or does not have permission to use this interface.

Postconditions

None

2.4.3 GL Retrieval Interface Operation ::get_account()

```
Account get_account (
        in AccountId acc_id )
        raises ( BadAccountId, PermissionDenied );
```

Description

This operation invocation retrieves GL Account summary information for a single, specific GL Account reference. If the operation was successful, the GL client has access to the **Account** data type, which includes the GL Account reference, name, and current balance. See the GL Retrieval interface operation **get_multiple_accounts()**, which may be used to retrieve a specified collection of GL Accounts.

Preconditions

None

Input Parameters

The **acc_id** parameter is passed in the invocation, which uniquely specifies the target GL Account in the currently selected General Ledger. The data type **AccountId** is a wide string data type.

Output Parameters

None

Return Value

This operation invocation returns the data type **Account,** which represents summary information about the specified GL Account in the currently selected General Ledger.

Exceptions

The exception BadAccountId will be raised if the specified GL Account is not found in the currently selected General Ledger or is invalid in some other way. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.4.4 GL Retrieval Interface Operation ::get_multiple_accounts()

```
AccountList get_multiple_accounts (
                in AccountIdList account_ids )
                raises ( BadAccountId, PermissionDenied );
```

Description

This operation invocation retrieves selected GL Account summary information for a specific collection of GL Accounts. If the operation was successful, the GL client has access to the **AccountList** data type, which includes the GL Account references, names and current balances of each of the specified GL Accounts in the currently selected General Ledger.

Preconditions

None

Input Parameters

The parameter **account_ids** is passed in the invocation, which specifies the target GL Accounts in the currently selected General Ledger.

Output Parameters

None

Return Value

This operation invocation returns an **AccountList** data type for the specified GL Accounts in the currently selected General Ledger.

Exceptions

The exception BadAccountId will be raised if any of the GL Account references specified is not found in the currently selected General Ledger or is invalid in some other way. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.4.5 GL Retrieval Interface Operation ::get_all_accounts()

AccountList get_all_accounts() raises (PermissionDenied);

Description

This operation invocation retrieves GL Account summary information for all GL Accounts in the currently selected General Ledger. If the operation was successful, the GL client has access to the **AccountList** data type, which includes the GL Account references, names, and current balances for all GL Accounts in the currently selected General Ledger.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

This operation invocation returns GL Account summary information for all GL Accounts in the currently selected General Ledger in the form of an **AccountList** data type.

Exceptions

The exception PermissionDenied is raised if the GL client has not successfully established a valid GL client session with **Arbitrator::open_session()** prior to the call or does not have permission to use this interface.

Postconditions

None

2.4.6 GL Retrieval Interface Operation ::get_trans_ids()

```
TransactionIdList get_trans_ids(
                in DateRange date_range)
                raises ( BadDate, PermissionDenied );
```

Description

This operation invocation retrieves the GL Transaction identifiers of all GL Transactions specified by an inclusive date range. If the operation was successful, the GL client has access to the **TransactionIdList** data type, which represents a collection type of GL Transaction references.

Preconditions

None.

Input Parameters

The parameter **date_range** is passed in the invocation, which specifies an inclusive start and end date, specifying the period within which GL Transactions should be retrieved.

Output Parameters

None

Return Value

This operation invocation returns a **TransactionIdList** data type, which is a sequence of TransactionId data types which in turn corresponds to the GL Transactions retrieved within the specified inclusive date range in the currently selected General Ledger.

Exceptions

The exception BadDate will be raised if either or both of the dates are deemed to be invalid. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.4.7 GL Retrieval Interface Operation ::get_trans_info_summary()

```
TransactionInfo get_trans_info_summary(
    in TransactionId trans_id)
        raises ( BadTransId, PermissionDenied );
```

Description

This operation invocation retrieves summary information about a single GL Transaction in the currently selected General Ledger, given a single, specific GL Transaction reference. If the operation was successful, the GL client has access to the **TransactionInfo** data type, which represents summary information about the specified GL Transaction.

Preconditions

None

Input Parameters

The parameter **trans_id** is passed in the invocation, which specifies a single GL Transaction reference.

Output Parameters

None

Return Value

This operation invocation returns a **TransactionInfo** data type that corresponds to a specific GL Transaction found within the currently selected General Ledger. The **TransactionInfo** data type includes summary information about the specified GL Transaction.

Exceptions

The exception BadTransId will be raised if the reference could not be located or was in some other way deemed to be invalid. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.4.8 GL Retrieval Interface Operation ::get_trans_info_by_refs()

```
TransactionInfoList get_trans_info_by_refs(
```

```
in TransactionIdList trans_ids)
raises ( BadTransId, PermissionDenied );
```

Description

This operation invocation retrieves multiple GL Transaction summaries for a specified list of GL Transaction references in the currently selected General Ledger. If the operation was successful, the GL client has access to the **TransactionInfoList** data type, which is a collection type for GL Transaction summaries.

Preconditions

None

Input Parameters

The parameter **trans_ids** is passed in the invocation, which precisely specifies the collection of GL Transaction references from which GL Transaction summary information should be retrieved.

Output Parameters

None

Return Value

This operation invocation returns a **TransactionInfoList** that is a **sequence** of **TransactionInfo** data types, which in turn correspond to GL Transactions found within the specified selection criteria in the currently selected General Ledger. The **TransactionInfo** data type includes summary information about the specified GL Transaction.

Exceptions

The exception BadTransId will be raised if any of the GL Transaction references specified are not found in the currently selected General Ledger or are invalid in some other way. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.4.9 GL Retrieval Interface Operation ::get_trans_info_by_date()

```
TransactionInfoList get_trans_info_by_date(
                    in DateRange date_range)
                    raises ( BadDate, PermissionDenied );
```

Description

This operation invocation retrieves all GL Transaction summaries for each GL Account in the currently selected General Ledger according to a specified inclusive date range. If the operation was successful, the GL client has access to the **TransactionInfoList** data type, which is a collection type for GL Transaction summaries within the specified date range.

Preconditions

None

Input Parameters

The parameter **date_range** is passed in the invocation that specifies an inclusive start and end date, specifying the period within which GL Transactions should be retrieved.

Output Parameters

None

Return Value

This operation invocation returns a **TransactionInfoList** that is a sequence of **TransactionInfo** data types that correspond to GL Transactions found within the specified date range in the currently selected General Ledger. The **TransactionInfo** data type includes summary information about the specified GL Transaction.

Exceptions

The exception BadDate will be raised if either or both of the dates are deemed to be invalid. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None.

2.4.10 GL Retrieval Interface Operation ::get_entry_count_by_account()

```
unsigned long get_entry_count_by_account(
                in DateRange date_range,
                in AccountId acc_id)
                raises (BadDate, BadAccountId, PermissionDenied );
```

Description

This operation invocation retrieves the total number of all GL Entries (i.e., individual GL Entries) based on a specified inclusive date range, given a single GL Account reference. The purpose of this operation is to assist clients in anticipating a possibly very large number of GL Entries for any particular GL Account. If the operation was successful, the

total number of GL Entries can be determined so that particular situations can be anticipated, where the number of GL Entries for a specific GL Account could be too large to handle. See Section 2.3.9, “GL Profile Interface Operation ::get_entry_types(),” on page 2-23 and Section 2.6.6, “GL LedgerLifecycle Interface Operation ::set_entry_types(),” on page 2-45 for more information.

Preconditions

None

Input Parameters

The parameter **date_range** is passed in the invocation, which specifies an inclusive start and end date. The parameter **acc_id** is also passed in the invocation, which specifies a single GL Account.

Output Parameters

None

Return Value

This operation returns an **unsigned long** that represents the total number of GL Entries belonging to a specific GL Account reference, which match the selection criteria in the currently selected General Ledger. Please note that the count returned by this operation is a minimum value. Other GL clients using the same General Ledger may have posted additional GL Transactions and GL Entries before a GL client makes use of the returned value.

Exceptions

The exception BadDate will be raised if either or both of the dates are deemed to be invalid. The exception BadAccountId will be raised if the specified GL Account reference is invalid or not found. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.4.11 GL Retrieval Interface Operation ::get_entry_count_by_type()

```
unsigned long get_entry_count_by_type(
                in DateRange date_range,
                in EntryType entry_type
                raises ( BadDate, BadEntryType, PermissionDenied)
```

Description

This operation invocation retrieves the total number of GL Entries (i.e., individual GL Entries) based on a specified inclusive date range, given a single GL Entry type. The purpose of this operation is to assist clients in anticipating a possibly very large number of GL Entries of any particular GL Entry type. If the operation was successful, the total number of GL Entries of a specific type can be determined so that particular situations can be anticipated, where the number of GL Entries could be too large to handle. See Section 2.3.9, “GL Profile Interface Operation ::get_entry_types(),” on page 2-23 and Section 2.6.6, “GL LedgerLifecycle Interface Operation ::set_entry_types(),” on page 2-45 for more information.

Preconditions

None

Input Parameters

The parameter **date_range** is passed in the invocation that specifies an inclusive start and end date. The parameter **entry_type** is also passed in the invocation that specifies a single GL Entry type.

Output Parameters

None

Return Value

This operation returns an **unsigned long** that represents the total number of GL Entries of the specified GL Entry type, which match the selection criteria in the currently selected General Ledger. Please note that the count returned by this operation is a minimum value. Other GL clients using the same General Ledger may have posted additional GL Transactions and GL Entries before a GL client makes use of the returned value.

Exceptions

The exception BadDate will be raised if either or both of the dates are deemed to be invalid. The exception BadEntryType will be raised if the specified GL Entry Type is invalid or not found. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.4.12 GL Retrieval Interface Operation ::get_transaction()

```
Transaction get_transaction(
        in TransactionId trans_id)
        raises ( BadTransId, PermissionDenied );
```

Description

This operation invocation retrieves a single GL Transaction summary and a list of associated GL Entries for a single, specific GL Transaction reference in the currently selected General Ledger. If the operation was successful, the GL client has access to a Transaction data type that represents summary information about the specified GL Transaction and associated GL Entries.

Preconditions

None

Input Parameters

The parameter **trans_id** is passed in the invocation that specifies the single, specific GL Transaction to retrieve from the currently selected General Ledger.

Output Parameters

None

Return Value

This operation invocation returns a **Transaction** data type that corresponds to a specific GL Transaction found within the currently selected General Ledger. The **Transaction** data type represents summary information about the specified GL Transaction and associated GL Entries. The Transaction data type also specifies an **EntryList** data type, which corresponds to the GL Entries associated with the specified GL Transaction.

Exceptions

The exception BadTransId will be raised if the specified GL Transaction reference could not be located or was in some other way deemed to be invalid. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.4.13 GL Retrieval Interface Operation ::get_transactions_by_ids()

```
TransactionList get_transactions_by_ids(
                    in TransactionIdList trans_ids)
                    raises ( BadTransId, PermissionDenied );
```

Description

This operation invocation retrieves a list of GL **Transaction** summaries and associated GL Entries for each GL **Transaction** reference in the currently selected General Ledger, given a list of GL Transaction references. If the operation was successful, the GL client

has access to a **TransactionList** data type, which is a collection of type GL **Transaction**. The GL **Transaction** data type represents summary information about the specified GL Transaction and associated GL Entries.

Preconditions

None

Input Parameters

The parameter **trans_id** is passed in the invocation that specifies a collection of GL Transaction references.

Output Parameters

None

Return Value

This operation invocation returns a **TransactionList** data type that specifies a collection of type Transaction. The **Transaction** data type includes information about a specified GL Transaction such as a Transaction reference as well as the physical and logical dates associated with the specified list of GL Transactions. The **Transaction** data type also specifies an **EntryList** data type, which corresponds to the GL Entries associated with a specific GL Transaction.

Exceptions

The exception BadTransId will be raised if a GL Transaction reference could not be located or was in some other way deemed to be invalid. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.4.14 GL Retrieval Interface Operation ::get_transactions_by_date()

```
TransactionList get_transactions_by_date(
                    in DateRange date_range)
                    raises ( BadDate, PermissionDenied );
```

Description

This operation invocation retrieves a list of all GL Transaction summaries and associated GL Entries for each GL Transaction in the currently selected General Ledger, which matches the input date parameters. If the operation was successful, the GL client has access to the **TransactionList** data type that specifies a collection of type Transaction.

Preconditions

None

Input Parameters

The parameter **date_range** is passed in the invocation, which specifies an inclusive start and end date.

Output Parameters

None

Return Value

This operation invocation returns a **TransactionList** data type that specifies a collection of type Transaction. The **Transaction** data type includes information about a specified GL Transaction such as a Transaction reference as well as the physical and logical dates associated with the specified list of GL Transactions. The **Transaction** data type also specifies an **EntryList** data type, which corresponds to the GL Entries associated with a specific GL Transaction.

Exceptions

The exception BadDate will be raised if the date is deemed to be invalid. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.4.15 GL Retrieval Interface Operation ::get_entries_by_type()

```
EntryList get_entries_by_type (
        in DateRange date_range,
        in EntryType entry_type)
        raises ( BadDate, BadEntryType, PermissionDenied );
```

Description

This operation invocation retrieves a list of all GL Entries (i.e., individual GL Entries) based on an inclusive date range and a single **EntryType** data type. If the operation was successful, the GL client has access to all GL Entries that match the specified **EntryType** that were posted in the specified date range. See Section 2.3.9, “GL Profile Interface Operation ::get_entry_types(),” on page 2-23 for more information.

Preconditions

None

Input Parameters

The parameter **date_range** is passed in the invocation that specifies an inclusive start and end date, as well as a specific GL **entry_type** identifying the type of GL Entries to retrieve.

Output Parameters

None

Return Value

This operation invocation returns an **EntryList** data type, which is a collection type for GL Entries.

Exceptions

The exception BadDate will be raised if either of the dates in the date range are deemed to be invalid. The exception BadEntryType will be raised if the **EntryType** parameter is invalid. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.4.16 GL Retrieval Interface Operation ::get_entries_by_account()

```
EntryList get_entries_by_account (
        in DateRange date_range,
        in AccountId acc_id)
        raises ( BadDate, BadAccountId, PermissionDenied );
```

Description

This operation invocation retrieves the number of all GL Entries (i.e., individual GL Entries) based on a date range and a single GL Account reference. If the operation was successful, the GL client has access to an **EntryList** data type of all GL Entries associated with a specific GL Account that were posted within the specified date range.

Preconditions

None

Input Parameters

The parameter **date_range** is passed in the invocation, which specifies an inclusive start and end date, specifying the period within which GL Entries should be retrieved as well as a specific GL Account reference that specifies the GL Entries to retrieve.

Output Parameters

None

Return Value

This operation invocation returns an **EntryList** data type, which represents a collection of GL Entries that match the date selection criteria and belong to the specified GL Account reference.

Exceptions

The exception BadDate will be raised if either of the dates in the specified date range are deemed to be invalid. The exception BadAccountId will be raised if the GL Account reference is invalid or not found. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

None

2.5 GL Facility - BookKeeping Interface

The GL Bookkeeping interface is used for entering new GL Transactions in the currently selected General Ledger.

<<Interface>>

BookKeeping

+ post_transaction (single_transaction : Transaction) : TransactionId

+ post_transaction_list (transactions : TransactionList) : TransactionIdList

2.5.1 GL BookKeeping Interface General Invariants

This section details invariants that are generally applicable to all operations supported by the GL BookKeeping interface.

A GL client session must have been established with the **Arbitrator::open_session()** operation and each GL client session must use a unique instance of the Profile interface. The GL Facility must have been pre-configured for general purpose usage and there must also be at least one General Ledger in existence for these operations to succeed. Furthermore, a non-empty Chart of Accounts must also exist. See Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2 and Section 2.1.4, “GL Facility - General Invariants,” on page 2-2 for more information.

2.5.2 GL BookKeeping Interface Operation ::post_transaction()

TransactionId post_transaction(

in Transaction single_transaction) raises (BadTransaction, PermissionDenied);

Description

This operation posts a single GL Transaction to the currently selected General Ledger. A GL Transaction is not only composed of information about the GL Transaction itself, but at a more fundamental level, a single GL Transaction is composed of a set of at least two (or more) GL Accounts and at least two (or more) GL Entries. A single GL Entry specifies a single GL Account in the currently selected General Ledger's Chart of Accounts, which will either be debited or credited.

In the context of a GL Transaction, a single GL Account may be conceptualized as either a source GL Account (debit) or a destination GL Account (credit). The GL Transaction ensures that money is always debited from the single specified GL Account (source).

The source GL Account is specified by a single GL Entry (which is a member of the set to be posted and constrained by the GL Transaction) and is called the "debit account." The GL Transaction also ensures that the money debited from the source GL Account is credited to another (different) GL Account (destination). The destination GL Account is specified by a single GL Entry, which is a member of the set constrained by the GL Transaction and is called the "credit account."

Preconditions

None

Input Parameters

The parameter **single_transaction** is passed in the invocation. The Transaction data type specifies a collection type for a single GL Transaction and its associated GL Entries.

Output Parameters

None

Return Value

This operation invocation returns a **TransactionId** data type, which specifies a unique GL Transaction reference for a single specific GL Transaction.

Exceptions

The exception BadTransaction will be raised if either the GL Transaction or its associated GL Entries have illegal or otherwise invalid values, or if the GL Entries constrained by the GL Transaction are not balanced. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

The new GL Transaction and its associated GL Entries are posted to the currently selected General Ledger, and the balances of each of the specified GL Accounts referenced by the GL Entries are updated accordingly.

2.5.3 GL BookKeeping Interface Operation ::post_transaction_list()

TransactionIdList post_transaction_list(
in TransactionList transactions)
raises (BadTransactionsInList, PermissionDenied);

Description

This operation invocation posts a collection of GL Transactions to the currently selected General Ledger. See the corresponding Section2.5.2, “ GL BookKeeping Interface Operation ::post_transaction(),” on page 2-38 for more information.

Preconditions

None

Input Parameters

The parameter transaction is passed in the invocation. The data type **TransactionList** specifies a collection type for multiple GL Transactions and the associated GL Entries.

Output Parameters

None

Return Value

This operation invocation returns a **TransactionIdList** data type, which specifies a collection type for GL Transaction references.

Exceptions

The exception BadTransactionsInList will be raised if any of the specified GL Transactions or their GL Entries have illegal or otherwise invalid values, or if any of the sets of GL Entries constrained by the GL Transactions are not balanced. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

The new GL Transactions are added to the currently selected General Ledger, and the balances of the GL Accounts referenced by the GL Entries are updated accordingly.

2.6 GL Facility - LedgerLifecycle Interface

The GL **LedgerLifecycle** interface is primarily intended for use in the establishment and subsequent maintenance of a specific Chart of Accounts for a specific General Ledger constrained and managed by the GL Facility. Generally, these operations are only used by privileged GL Facility Administrators. See Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2 and Section 2.1.4, “GL Facility - General Invariants,” on page 2-2 for more information.

The GL **LedgerLifecycle** interface operations support the creation and subsequent revision of a specified General Ledger’s Chart of Accounts, which was initialized with the **EMPTY_CHART** member of the **ChartKind** data type, by allowing for the addition, removal, and modification of specified GL Accounts. See Section 2.1.12, “GL Facility - Basic Data Type Information,” on page 2-6.

The GL **LedgerLifecycle** interface may also be used to establish the reporting currency for a specific General Ledger, as well as establishing or revising a set of GL Entry types used to identify the **EntryType** of GL Entry. See Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2 and Section 2.1.4, “GL Facility - General Invariants,” on page 2-2 for more information.

<<Interface>>

LedgerLifecycle

+ create_account (acc_id : AccountId, is_control_account : boolean, acc_description : wstring) : void

+ remove_account (acc_id : AccountId) : void

+ modify_account (acc_id : AccountId, new_acc_description : wstring) void

+ set_ledger_currency (currency_mnemonic : CurrencyMnemonic) : void

+ set_entry_types (entry_types : EntryTypeInfoList) : void

2.6.1 GL LedgerLifecycle Interface General Invariants

This section details invariants that are generally applicable to all operations supported by the GL **LedgerLifecycle** interface.

A GL client session must have been established with the **Arbitrator::open_session()** operation and each GL client session must use a unique instance of the **Profile** interface. The GL Facility must have been pre-configured for general purpose usage and there must also be at least one General Ledger in existence for these operations to succeed. Furthermore, a non-empty Chart of Accounts must also exist. See Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2 and Section 2.1.4, “GL Facility - General Invariants,” on page 2-2 for more information.

2.6.2 GL LedgerLifecycle Interface Operation ::create_account()

```
void create_account(
        in AccountId acc_id,
```

```
    in wstring acc_description,
    in boolean is_control_account)
    raises (BadAccountId, BadAccountName, PermissionDenied );
```

Description

This operation adds a new GL Account to the currently selected General Ledger's Chart of Accounts schema. This is an infrequently used operation and is typically only used by privileged GL Administrators.

Preconditions

None

Input Parameters

The parameter **acc_id** is passed in the invocation that specifies a unique reference to the new GL Account to be created. The parameter **acc_description** is also passed in the invocation, which specifies a high level name describing the purpose to which the new account will be used (for example, salaries, bank current account). The parameter **is_control_account** is also passed in the invocation, which determines whether the new account will behave as a "Control Account." A GL Control Account accumulates a monetary value of related subsidiary GL Accounts over time. Some examples of GL Control Account designations could be Debtor's Control Account, Creditor's Control Account, Inventory, Buildings and Equipment.

Output Parameters

None

Return Value

None

Exceptions

The exceptions BadAccountId and BadAccountName will be raised if either or both of the input parameters are unknown or are otherwise invalid. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

The new GL Account is created and appears in the Chart of Accounts of the currently selected General Ledger.

2.6.3 GL LedgerLifecycle Interface Operation ::remove_account()

```
void remove_account(
    in AccountId acc_id)
    raises ( BadAccountId, CannotRemove, PermissionDenied );
```

Description

This operation removes a single, specific GL Account from the currently selected General Ledger's Chart of Accounts schema. This is an infrequently used operation and is typically only used by privileged GL Administrators.

Preconditions

None

Input Parameters

The parameter **acc_id** is passed in the invocation, which explicitly specifies the GL Account to be removed.

Output Parameters

None

Return Value

None

Exceptions

The exception BadAccountId will be raised if the reference cannot be located in the currently selected General Ledger or is otherwise deemed invalid. The exception CannotRemove will be raised if an unauthorized attempt is made to remove a GL Account. See Section 2.1.3, "GL Facility - Environment Contract," on page 2-2 and Section 2.1.4, "GL Facility - General Invariants," on pag e2-2 for more information. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

The GL Account specified by the data type **AccountId** passed in the invocation is removed from the currently selected General Ledger's Chart of Accounts schema.

2.6.4 GL LedgerLifecycle Interface Operation ::modify_account()

```
void modify_account(
        in AccountId acc_id,
        in wstring new_acc_description)
        raises ( BadAccountId, BadAccountName, PermissionDenied );
```

Description

This operation modifies only the descriptive name associated with the specified GL Account.

Preconditions

None

Input Parameters

The parameter **acc_id** is passed in the invocation, which explicitly specifies the GL Account to modify, as well as a new account description for the specified GL Account (for example, salaries or bank current account).

Output Parameters

None

Return Value

None

Exceptions

The exception BadAccountId will be raised if the GL Account reference cannot be located in the currently selected General Ledger or is otherwise deemed to be invalid. The exception BadAccountName will be raised if the information specified by this parameter is invalid or the specified GL Account description already exists. The exception PermissionDenied will be raised if the GL client does not have permission to invoke this interface operation.

Postconditions

The specified GL Account name has been modified as specified.

2.6.5 GL LedgerLifecycle Interface Operation ::set_ledger_currency()

```
void set_ledger_currency (
    in CurrencyMnemonic currency_mnemonic)
    raises ( BadCurrencyMnemonic, PermissionDenied );
```

Description

This operation specifies the reporting currency of the currently selected General Ledger. This interface operation is typically used when setting up a new General Ledger. See Section 2.3.8, “GL Profile Interface Operation ::get_ledger_currency(),” on page2-22 for more information on the reporting currency, as well as Appendix A.

Preconditions

None

Input Parameters

The parameter **currency_mnemonic** is passed as a parameter in the invocation.

Output Parameters

None

Return Value

None

Exceptions

The exception BadCurrencyMnemonic is raised if there is a problem with the input parameter. The exception PermissionDenied is raised if it is deemed that the GL client does not have permission to set the specified General Ledger's reporting currency.

Postconditions

The currently selected General Ledger's reporting currency is set as specified in the invocation.

2.6.6 GL LedgerLifecycle Interface Operation ::set_entry_types()

```
void set_entry_types(
    in EntryTypeInfoList entry_types)
    raises ( BadEntryTypeInfoList, PermissionDenied );
```

Description

This operation is used to establish or revise a set of GL Entry types. Please see the corresponding GL Profile operation **set_entry_types()** for detailed information.

Preconditions

None

Input Parameters

The parameter **entry_types** is passed in the invocation that specifies name/value pairs describing both the purpose of the GL Entry **EntryType** as well as their mnemonic identifiers.

Output Parameters

None

Return Value

None

Exceptions

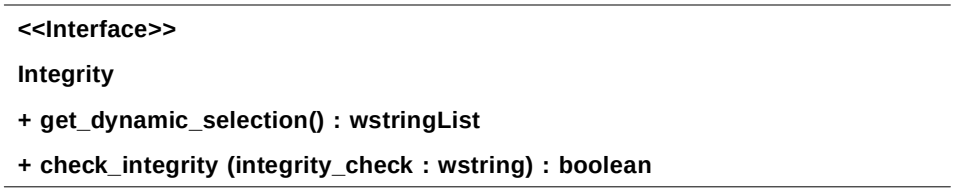
The exception BadEntryTypeInfoList is raised if there is a problem with the input parameter passed in the invocation. The exception PermissionDenied is raised if it is deemed that the GL client does not have permission to set the specified General Ledger's GL Entry types.

Postconditions

The currently selected General Ledger's GL Entry types are set as specified in the invocation.

2.7 GL Facility - Integrity Interface

The GL Integrity interface provides for general purpose integrity checks on a specific General Ledger's Chart of Accounts, Transactions and Entries in the currently selected General Ledger. Generally, these operations are only used by privileged GL Facility Administrators. See Section 2.1.3, "GL Facility - Environment Contract," on page 2-2 and Section 2.1.4, "GL Facility - General Invariants," on page 2-2 for more information.



Although a specific underlying technical infrastructure may offer certain assurances with regard to general *data* integrity, in practice, it is not uncommon that the necessity arises to verify and test the integrity of specific *information*. Even though this specification provides for the flexible retrieval of GL related information, there are numerous common situations that implementations claiming conformance with this specification could provide to GL clients that need to "troubleshoot" GL information, either pre-emptively or retrospectively.

For example, implementations claiming conformance with this specification could check that the current balance of a specified GL Account (or set of GL Accounts) is in agreement with the total reported after the summation of associated GL Transactions and Entries, or perhaps check that there are no "phantom" GL Transactions (i.e., GL Transactions that for some unknown reason have no GL Entries associated with them).

2.7.1 GL Integrity General Invariants

This section details invariants that are generally applicable to all operations supported by the GL Integrity interface.

A GL client session must have been established with the **Arbitrator::open_session()** operation and each GL client session must use a unique instance of the **Profile** interface. The GL Facility must have been pre-configured for general purpose usage and there must also be at least one General Ledger in existence for these operations to succeed.

Furthermore, a non-empty Chart of Accounts must also exist. See Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2 and Section 2.1.4, “GL Facility - General Invariants,” on page 2-2 for more information.

The GL Facility makes available a non-zero list of supported GL Integrity checks that may be used by suitably authorized GL clients. This specification does not mandate the specific nature or number of available integrity checks but does mandate that implementations claiming conformance with this specification support GL Integrity checking.

2.8 GL Integrity Interface Operation

2.8.1 GL Integrity Interface Operation ::get_dynamic_selection()

wstringList get_dynamic_selection()raises (PermissionDenied);

Description

This operation invocation returns a **wstringList** of available integrity checks to be performed on the information contained in the currently selected General Ledger. Each integrity check is represented by a unique human-readable name, which specifies both the purpose and scope of the integrity check to be performed.

Preconditions

None

Input Parameters

None

Output Parameters

None

Return Value

This operation invocation returns a **wstringList** containing an implementation defined list of integrity checks.

Exceptions

The exception PermissionDenied is raised if the GL client has not successfully established a valid GL client session with **Arbitrator::open_session()** prior to the call or does not have permission to use this interface.

Postconditions

None

2.8.2 GL Integrity Interface Operation ::check_integrity()

```
boolean check_integrity (
        in wstring integrity_check)
        raises ( BadIntegritySelection, PermissionDenied );
```

Description

This operation is used to verify the integrity of the General Ledger data in the currently selected General Ledger. The specific integrity check to be performed on the currently selected General Ledger must be one of those returned by the **get_dynamic_selection()** operation, which has to be invoked first.

Preconditions

None

Input Parameters

The parameter **integrity_check** is passed in the operation invocation.

Output Parameters

None

Return Value

This operation invocation returns TRUE if the integrity check passes or FALSE if there are warnings, errors, or other inconsistencies detected.

Exceptions

The exception BadIntegritySelection will be raised if the name of the specified integrity check is invalid or unknown to the GL Facility. The exception PermissionDenied is raised if it is deemed that the GL client does not have permission to invoke this operation.

Postconditions

The specified GL integrity check either succeeded or failed.

2.9 GL Facility - FacilityLifecycle Interface

The GL **FacilityLifecycle** operations are intended for use by GL client sessions that are oriented toward GL administrative functionality. Generally, these operations are only used by privileged GL Facility Administrators. See Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2 and Section 2.1.4, “GL Facility - General Invariants,” on page 2-2 for more information.

These operations are used to initially establish a Chart of Accounts schema for a specific General Ledger constrained and managed by the GL Facility, as well as providing the capability to remove a specified General Ledger from a GL Facility.

<<Interface>>

FacilityLifecycle

+ create_ledger_chart_of_accounts (new_ledger_name : wstring,
chart_of_accounts_schema : ChartKind,
copied_ledger_name_for_schema : wstring) : void

+ remove_ledger (ledger_name : wstring) : void

2.9.1 GL FacilityLifecycle Interface General Invariants

This section details invariants that are generally applicable to all operations supported by the GL **FacilityLifecycle** interface.

A GL client session must have been established with the **Arbitrator::open_session()** operation and each GL client session must use a unique instance of the **Profile** interface. To remove a specific General Ledger, the GL Facility must have been pre-configured for general purpose usage and there must also be at least one General Ledger in existence for this operation to succeed. See Section 2.1.3, “GL Facility - Environment Contract,” on page 2-2 and Section 2.1.4, “GL Facility - General Invariants,” on page 2-2 for more information.

2.9.2 GL FacilityLifecycle Interface Operation ::create_ledger_chart_of_accounts()

```
void create_ledger_chart_of_accounts (
    in wstring new_ledger_name,
    in ChartKind chart_of_accounts_schema,
    in wstring copied_ledger_name_for_schema )
    raises ( UnknownLedger, BadChartKind, PermissionDenied );
```

Description

This operation is used to create a new General Ledger, to be constrained and managed by an instance of the GL Facility. Please note that this is an infrequently used operation and is typically only carried out by privileged GL Facility Administrators. See Sect ion2.1.12, “GL Facility - Basic Data Type Information,” on page 2-6.

Preconditions

None

Input Parameters

The parameter **new_ledger_name** is passed in the invocation that specifies the unique name of the new General Ledger to create.

The parameter **chart_of_accounts_schema** is also passed in the invocation that specifies whether to create an **EMPTY_CHART** of GL Accounts (i.e., zero GL Accounts) or a **DEFAULT_CHART** of GL Accounts, or whether the GL Facility implementation should create a "clone" from an **EXISTING_CHART** of GL Accounts in an existing General Ledger known to the GL Facility.

If the **chart_of_accounts_schema** specifies that an existing Chart of Accounts in another General Ledger should be used as a model and, if the **copied_ledger_name_for_schema** specifies a valid General Ledger name known to the GL Facility, then the Chart of Accounts schema is copied from the specified General Ledger constrained and managed by the GL Facility.

Output Parameters

None

Return Value

None

Exceptions

The exception UnknownLedger is raised if the new Chart of Accounts schema is to be based on another General Ledger and the specified General Ledger cannot be located or is otherwise invalid.

The exception BadChartKind is raised if an invalid Chart of Accounts kind is specified.

The exception PermissionDenied is raised if it is deemed that the GL client does not have permission to invoke this interface operation.

Postconditions

A new General Ledger is created and is made generally available for new GL client sessions and all GL Account balances in the new General Ledger are set to zero. The new General Ledger appears in the **wstringList** returned by the **Profile::get_ledger_names()** operation.

2.9.3 GL FacilityLifecycle Interface Operation ::remove_ledger()

```
void remove_ledger (
    in wstring ledger_name )
    raises ( UnknownLedger, CannotRemove, PermissionDenied );
```

Description

This operation removes a single, specific General Ledger from the GL Facility. See Section 2.1.3, “GL Facility - Environment Contract,” on page2-2 and Section2.1.4, “ GL Facility - General Invariants,” on page 2-2 for more information.

See Section 2.2.2, “GL Arbitrator Interface Operation ::get_ledger_names(),” on page 2-14 for more information on the retrieval of General Ledger names constrained and managed by an instance of the GL Facility.

Preconditions

None

Input Parameters

The parameter **ledger_name** is passed in the operation invocation that refers to the name of a single, specific General Ledger constrained and managed by the GL Facility. This parameter explicitly specifies the General Ledger to remove.

Output Parameters

None

Return Value

None

Exceptions

The exception UnknownLedger is raised if the new Chart of Accounts schema is to be based on another General Ledger and that General Ledger cannot be located or is otherwise invalid. The exception CannotRemove will be raised according to specific organizational policy decisions. The exception PermissionDenied is raised if it is deemed that the GL client does not have permission to invoke this interface operation.

Postconditions

The specified General Ledger has been removed from the GL Facility.

Complete OMG IDL A

A.1 Explanation of IDL Changes

The architectural intention of the GL Facility's API design is to facilitate the convenient implementation of interoperable accounting applications, referred to as "clients" in the specification. The overall intention is to provide as complete a set of GL services as possible to support the implementation of accounting clients that need to interoperate with one or more GL Facility implementations. One or more GL client(s) makes a request for a new GL client session from the GL Facility's Arbitrator interface. If successful, the GL Arbitrator returns a GL Profile interface that provides information about the current session, as well as making provision for controlled access to the various interfaces and operations supported by the GL Facility.

During the evaluation process, discussion always centered around the **login_name** and password arguments of the GL Arbitrator's **open_session()** operation, and whether or not these should be supplied by the CORBA Security service or the facility itself, if no CORBA Security service was available. The authors took this approach to help the FDTF to determine "policy" for the future evaluation of other proposed FDTF Domain Interface specifications. The authors always had an open mind as to whether or not these arguments should be included in the specification, but the final decision should be based on a consensus between FDTF, the CORBAsecurity SIG and others. Consensus was finally reached that these arguments should be excluded from the specification, hence the change to the IDL as detailed in this document.

A.2 General Ledger Facility IDL

```
#ifndef _FdGeneralLedger_
#define _FdGeneralLedger_

/*
 * GENERAL LEDGER FACILITY IDL
 *
 * Revision History as of October 1998.
```

```
 * 20-10-1998 Additions to Revised Submission - York.
 * 30-10-1998 Revision and Clarification - Manchester.
 * 31-10-1998 Logical corrections - Manchester.
 * 01-11-1998 Syntax errors corrected - Orbix 2.3 IDL compiler - Washington.
 * 09-11-1998 Additional items left out from York - Leamington Spa.
 * 10-11-1998 Minor revisions and clarifications - San Francisco.
 * 27-11-1998 co-submitters meeting - London.
 * 28-11-1998 through to 02-12-1998 - Manchester.
 * 03-12-1998 through to 06-12-1998 - Oslo.
 * 07-12-1998 through to 21-12-1998 - all co-submitters.
 *
*/

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

module FdGeneralLedger                  // FORWARD DECLARATIONS

    interface Arbitrator;               // establish client session
    interface Profile;                  // GL facility metadata
    interface Retrieval;                // data acquisition
    interface BookKeeping;              // data entry
    interface LedgerLifecycle;          // lifecycle management
    interface Integrity;                // data integrity checks
    interface FacilityLifecycle;        // Facility lifecycle management
                                        // DATA TYPE DECLARATIONS
    typedef FbcCurrency::Money Money;
    typedef sequence<wstring> wstringList;
    typedef wstring CurrencyMnemonic;   // ISO currency mnemonic
    typedef wstring TransactionId;      // a transaction reference
    typedef sequence<TransactionId> TransactionIdList;
    typedef wstring EntryId;
    typedef sequence<EntryId> EntryIdList;
    typedef wstring AccountId;
    typedef sequence<AccountId> AccountIdList;
    typedef wstring EntryType;
    typedef sequence<EntryType> EntryTypeInfoList;
    typedef wstring PeriodId;           // reference to a user defined
                                        // accounting period


    typedef unsigned short ChartKind;
                                        // used to select a kind of new
                                        // ledger schema
    const ChartKind EMPTY_CHART     = 1;
    const ChartKind DEFAULT_CHART   = 2;
    const ChartKind EXISTING_CHART  = 3;

    enum DebitOrCredit { CREDIT, DEBIT };   // an entry is either a credit or
                                            // a debit

    struct Date {                       // Used for all date fields
CBO::DTime setting;                     // Does not allow "not set" dates
boolean is_set;                         // Needed for any useful date
                                        // data type    };
```

```
    struct DateRange { Date start_date; Date end_date; };
                                                    // specifies date ranges for
                                                    // purposes of retrieval
    struct AccountInfo { AccountId acc_id;     // account reference
                wstring acc_description; // account name
    };
    typedef sequence<AccountInfo> AccountInfoList;

    struct Account {                                // metadata describing an account
        AccountInfo     acc_info;               // account reference and name
        boolean         is_control_account;
                                                    // control accounts maintain
                                                    // running balances
        Money           balance;                // accumulated balance as of
                                                    // "current" date
};
    typedef sequence<Account> AccountList;// List of accounts

    struct TransactionInfo {                        // Summary information about a
                                                    // transaction
TransactionId   trans_id;                       // Transaction reference
                wstring voucher_ref;            // documentation of transaction
                Date    voucher_date;           // Date of initiating action that
                                                    // causes the posting
                Date    act_trans_date;         // Date of transaction posting -
                PeriodIdperiod_id;              // used for retrieval purposes
    };
    typedef sequence <TransactionInfo> TransactionInfoList;

    struct Entry {                                  // GL accounting entry
                                                    // informationTransactionId
                trans_id;                       // This entry is part of this transaction
                EntryId entry_id;               // audit trail number for this entry
                Date    entry_date;             // Date of entry posting
                EntryTypeentry_type;            // e.g. JournalCredit, JournalDebit etc.
                AccountIdacc_id;                // reference
                Money   orig_amount;            // original amount in this currency
                Money   amount;                 // Quantity and currency units of this entry
                DebitOrCreditdr_cr;             // Debit or Credit GL Account
                wstring     description;        // annotation describing the entry
                wstring     voucher_ref;        // reference to documentation of entry
    };
    typedef sequence<Entry> EntryList;          // List of accounting entries
    struct Transaction {                            // GL accounting transaction information
                TransactionInfo trans_info;// Summary information about transaction
    EntryList   entries;                            // Entries comprising the transaction
    };
    typedef sequence<Transaction> TransactionList;
                                                    // List of transactions
    struct EntryTypeInfo {
                EntryTypetype;                  // mnemonic of GL Entry type
                wstring description; };         // Journal Debit, Credit etc
    typedef sequence<EntryTypeInfo> EntryTypeInfoList;
```

```
                                                    // EXCEPTION DECLARATIONS
        exception BadDate { wstring error; Date bad_value; };

        exception BadChartKind { wstring error; ChartKind bad_value; };

        exception BadTransaction { wstring error; };

        exception BadTransactionsInList {
            wstring error;
            unsigned long position_in_list; };

        exception BadEntryType {
            wstring error;
            EntryType bad_value; };

        exception BadEntryTypeInfoList {
            wstring error;
            EntryTypeInfoList bad_values; };

        exception BadCurrencyMnemonic {
            wstring error;
            CurrencyMnemonic bad_value; };

        exception BadAccountId {
            wstring error;
            AccountId bad_value; };

        exception BadTransId {
            wstring error;
            TransactionId bad_value; };

        exception CannotRemove { wstring error; };

        exception PermissionDenied { wstring error; };

        exception UnknownLedger {
            wstring error;
            wstring bad_value; };

        exception BadIntegritySelection {
            wstring error;
            wstring bad_value; };

        exception BadAccountName {
            wstring error;
            wstring bad_value; };

                                                    // INTERFACE DECLARATIONS    interface Arbitrator {/
                                                    // Initiate client session on a specified General Ledger.
            wstringList get_ledger_names();         // Returns names of available ledgers.
        Profile open_session (
            in wstring ledger_name,
            in wstring login_name, in wstring password)
            raises ( UnknownLedger, PermissionDenied );
};
```

```
interface Profile {

    void close_session();
                                        // End this client session, reclaiming
                                        // session resources and state

                                        // GL FRAMEWORK OPERATIONS

BookKeeping book_keeping() raises ( PermissionDenied );
        // Get bookkeeping object reference

Retrieval retrieval() raises ( PermissionDenied );
        // Get retrieval service reference

Integrity integrity() raises ( PermissionDenied );
        // Get integrity service reference

LedgerLifecycle ledger_lifecycle() raises ( PermissionDenied );
        // Get ledger lifecycle service reference

FacilityLifecycle facility_lifecycle() raises ( PermissionDenied );
        // Get facility lifecycle service reference

        // PROFILE INFORMATION

CurrencyMnemonic get_ledger_currency() raises ( PermissionDenied );
            // Principal currency used in current ledger
EntryTypeInfoList get_entry_types() raises ( PermissionDenied );
            // Retrieve list of defined entry types };
interface Retrieval {
            // ACCOUNT METADATA RETRIEVAL

AccountInfoList get_all_account_info();
            // Retrieve account information.

Account get_account ( in AccountId acc_id )
    raises ( BadAccountId, PermissionDenied );
            // Retrieve description of an account

AccountList get_multiple_accounts ( in AccountIdList account_ids )
    raises ( BadAccountId, PermissionDenied );
            // Retrieve description for several accounts

AccountList get_all_accounts() raises ( PermissionDenied );
            // Retrieve all account summaries

            // TRANSACTION AND ENTRY METADATA

TransactionIdList get_trans_ids(in DateRange date_range)
    raises ( BadDate, PermissionDenied );
            // Retrieve number and list of account
            // transaction references
            // in the specified data range
```

```
TransactionInfo get_trans_info_summary(inTransactionId trans_id)
                raises ( BadTransId, PermissionDenied );
                    // Retrieve a transaction summary
TransactionInfoList get_trans_info_by_refs(
    in TransactionIdList trans_ids)
    raises ( BadTransId, PermissionDenied );
                    // Retrieve a specific list of
                    // transaction references
TransactionInfoList get_trans_info_by_date(in DateRange date_range)
    raises (BadDate, PermissionDenied);
                    // Retrieve transaction summaries from
                    // the specified date range

unsigned long get_entry_count_by_type(
    in DateRange date_range,
    in EntryType entry_type)
    raises ( BadDate, BadEntryType, PermissionDenied );
                    // Retrieve the count of entries with
                    // the indicated entry type
                    // (within the specified date range)

unsigned long get_entry_count_by_account(in DateRange date_range,
    in AccountId acc_id)
    raises ( BadDate, BadAccountId, PermissionDenied );
                    // Retrieve the count of entries for
                    // an account within a date range
                // TRANSACTION AND ENTRY RETRIEVAL
Transaction get_transaction(in TransactionId trans_id)
    raises ( BadTransId, PermissionDenied );
                    // Retrieve a transaction info and its entries
TransactionList get_transactions_by_ids(
    in TransactionIdList trans_ids)
    raises ( BadTransId, PermissionDenied );
                    // Retrieve a list of transactions by
                    // reference number
TransactionList get_transactions_by_date(
    in DateRange date_range)
    raises ( BadDate, PermissionDenied );
                    // Retrieve a list of
                    // transactions with entries within a
                    // date range
EntryList get_entries_by_type (
    in DateRange date_range,
    in EntryType entry_type)
    raises ( BadDate, BadEntryType, PermissionDenied );
                    // Retrieve a list of entries with the
                    // indicated entry type

EntryList get_entries_by_account (
    in DateRange date_range,
    in AccountId acc_id)
    raises ( BadDate, BadAccountId, PermissionDenied );
                    // Retrieve a list of entries for an
                    // account (within a date range)    };
```

```
interface BookKeeping {
                    // Each transaction is balanced. The
                    // sum of debits and credits balance.
TransactionId post_transaction(
    in Transaction single_transaction)
    raises ( BadTransaction, PermissionDenied );
                    // Post a transaction to the GL
                    // persistently - including one or
                    // more related entries
TransactionIdList post_transaction_list(
    in TransactionList transactions)
    raises ( BadTransactionsInList, PermissionDenied );
                    // Post several transactions to the GL
                    // persistently - including one
                    // or more sets of related entries  };

interface LedgerLifecycle {
                    // GL ACCOUNT LIFECYCLE

void create_account(in AccountId acc_id,
    in wstring acc_description,
    in boolean is_control_account)
    raises ( BadAccountId, BadAccountName, PermissionDenied );
                    // add a new account to the
                    // ledger schema
void removeAccount(
    in AccountId acc_id)
    raises ( BadAccountId, CannotRemove, PermissionDenied );
                    // Remove an account description from
                    // the ledger schema

void modify_account(in AccountId acc_id,
    in wstring new_acc_description)
    raises ( BadAccountId, BadAccountName, PermissionDenied );
                    // Modify an account description

                    // LEDGER METADATA MANAGEMENT
void set_ledger_currency (
    in CurrencyMnemonic currency_mnemonic)
    raises ( BadCurrencyMnemonic, PermissionDenied );
                    // Specify the principal currency
                    // for this ledger
void set_entry_types(
    in EntryTypeInfoList entry_types)
    raises ( BadEntryTypeInfoList, PermissionDenied );
                    // Establish or revise a set of entry
                    // types    };
interface Integrity {
    wstringList get_dynamic_selection() raises ( PermissionDenied );
                    // Retrieve a list of integrity
                    // checking options
boolean check_integrity (
    in wstring integrity_check)
    raises ( BadIntegritySelection, PermissionDenied );
                    // Verify the integrity of the ledger
```

```
                    // data and/or metadata    };
    interface FacilityLifecycle {
                    // LEDGER/CHART OF ACCOUNTS LIFECYCLE
    void create_ledger_chart_of_accounts (
        in wstring new_ledger_name,
        in ChartKind chart_of_accounts_schema,
        in wstring copied_ledger_name_for_schema )
        raises ( UnknownLedger, BadChartKind, PermissionDenied );
                    // add a new ledger to this facility
                    // instance

    void remove_ledger (
        in wstring ledger_name )
        raises ( UnknownLedger, CannotRemove, PermissionDenied );
                    // Delete a ledger from this facility
                    // instance    };};
                    // end of module FdGeneralLedger
#endif
```

References B

B.1 List of References

1. Executive Encyclopedia: Fortune, 1987.
2. P. Allen and S. Frost, Component-Based Development for Enterprise Systems, Applying The SELECTIVE Perspective: Cambridge, 1998.
3. G. Booch, I. Jacobson, and J. Rumbaugh, “UML Semantics,” Rational Software Corporation Version 1.0, January 13 1997.
4. W. J. Brown, R. C. Malveau, H. W. M. III, and T. J. Mowbray, Anti Patterns, Refactoring Software, Architectures, and Projects in Crisis: John Wiley & Sons, Inc., 1998.
5. C. F. Cargill, Information Technology Standardization: Theory, Process and Organizations: Digital Press, 1989.
6. A. Cockburn, “Structuring Use Cases with Goals,” , 1997.
7. COMPASS, “COMPASS Software Engineering Handbook Part I - IV,” , 1998.
8. COMPASS, “Guide to Economics,” 1998.
9. COMPASS, “Volume I: Architecture Overview,” 1998.
10. COMPASS, “Volume II:,” 1998.
11. COMPASS, “Volume III,” 1998.
12. COMPASS, “Volume IV: GL Extensions and Components,” 1998.
13. COMPASS, “Volume V: Technology Viewpoint,” 1998.
14. M. Fowler, Analysis Patterns: Reusable Object Models: Addison-Wesley, 1997.

15. M. Fowler and K. Scott, UML distilled - applying the standard object modeling language": Addison Wesley, ISBN 0-201-32563, 1997.

16. A. S. Hollander, E. L. Denna, and J. O. Cherrington, Accounting, Information Technology, and Business Solutions: IRWIN, 1996.

17. IASC, "International Accounting Standards," 1998. International Accounting Standards Committee

18. Y. Ijiri, Management Goals and Accounting for Control, vol. 3. Amsterdam, Netherlands: North-Holland, 1965.

19. Y. Ijiri, Momentum Accounting and Triple-Entry Bookkeeping, vol. 31. Sarasota: American Accounting Association, 1989.

20. ISO/IEC, "JTC1/SC21 Open Systems Interconnection, Data Management and Open Distributed Processing," , USA (ANSI).

21. ISO/IEC, "ISO/IEC 10746-1 Information technology - Basic reference model of Open Distributed Processing - Part 1: Overview," ISO ITU-T X.901 - ISO/IEC DIS 10746-1, 1996.

22. ISO/IEC, "ISO/IEC 10746-2 Information technology - Open Distributed Processing - Reference Model:Foundations," , 1996.

23. ISO/IEC, "ISO/IEC 10746-3 Information technology - Open Distributed Processing - Reference Model: Architecture," , 1996.

24. I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, Object-Oriented Software Engineering - A Use Case Driven Approach: Addison-Wesley, 1992.

25. R. E. Jensen, Phantasmagoric Accounting, vol. 14. Sarasota: American Accounting Association, 1976.

26. H. Kilov, B. Rumpe, and I. Simmonds, "OOPSLA'97 Workshop on Object Oriented Behavioral Semantic," Institut fur Informatik der Technischen Universitat Munchen 1997.

27. H. Kilov, B. Rumpe, and I. Simmonds, "OOPSLA'98 Workshop on Behavioral Semantics of OO Business and System Specifications," Institut fur Informatik der Technischen Universitat Munchen 1998.

28. MAGMA, "Magma Software engineering handbook," SINTEF, draft 1997.

29. C. R. Malburg, Accounting for a new business: Bob Adams Inc, 1994.

30. L. v. Mises, Human: Action: A Treatise on Economics: Regnery, 1963.

31. T. J. Mowbray, "How to apply open systems to OO architectures," in OBJECT Magazine, 1996.

32. T. J. Mowbray and R. C. Malveau, CORBA Design Patterns: John Wiley & Sons, Inc., 1997.

33. T. J. Mowbray and W. A. Ruh, Inside CORBA: John Wiley & Sons, 1997.

34. T. J. Mowbray and R. Zahavi, The Essential CORBA: Systems Integration Using Distributed Objects: John Wiley & Sons, Inc., 1995.

35. OMG, “Common Facilities RFP3,” OMG Document Number 95.11.3, November 1995.

36. OMG, “CORBAfacilities: Common Facilities Architecture,” Object Management Group Revision 4.0, November 1995.

37. OMG, “OMG Object Management Architecture Guide (OMA Guide), Revision 3.0,” , 1995.

38. OMG, “CORBAservices: Common Object Services Specification,” , 1997.

39. OMG, “The Common Object Request Broker: Architecture and Specification, Revision 2.2,” Object Management Group Feb. 1998.

40. OMG/UML, “UML Notation,” . http://www.rational.com/uml/html/notation, 1997.

41. OMG/UML, “UML Semantics,” . http://www.rational.com/uml/html/semantics, 1997.

42. R. Orfali and D. Harkey, Client/Server Programming with Java and CORBA: John Wiley & Sons, Inc., 1997.

43. T. Reenskaug, P. Wold, and O. A. Lehne, Working with Objects - The OOram Software Engineering Method: Manning Publications, ISBN 1-884777-10-4, 1996.

44. J. D. Shank and V. Govindarajan, “Strategic Cost Analysis: The Crown Cork and Seal Case,” Journal of Cost Management, vol. 2, pp. pp 5-16, 1989.

45. J. D. Shank and V. Govindarajan, “Strategic Cost Management and the Value Chain,” Journal of Cost Management, vol. 5, pp. pp 5-21, 1992.

46. M. Shaw and D. Garlan, Software Architecture - Perspectives On An Emerging Discipline: Prentice-Hall, 1996.

47. J. Siegel, CORBA Fundamentals and Programming: John Wiley & Sons, 1997.

48. C. Szyperski, Component Software, Beyond Object-Oriented Programming: Addison-Wesley, 1998.

49. P. B. B. Turney, Common Cents: The ABC Performance Breakthrough: Hillsboro, 1991.

Requirements C

C.1 Proof of Concept Statement

The principal contributors to this specification are involved with the European Union funded COMPASS project, as part of which they have developed two alternative prototype commercial implementations of General Ledger software guided by extensive consultation with end users and vendors as well as utilizing existing OMG compliant technology. The two implementations are based on existing accounting products and demonstrate the applicability of this specification to both legacy and component-based software; they will be available for display at the OMG Technical Meetings following the final specification.

C.2 Service Dependencies and Relationships

C.2.1 OMG Security Service

This particular aspect of the specification has been the subject of a great deal co-operative work with the CORBA Security SIG, the FDTF and the co-submitters for a considerable length of time. Security is of paramount importance when dealing with highly sensitive financial information. The co-submitters and their supporters believe that the approach taken explicitly specifies a significant degree of support for application-level security capabilities (i.e., dealing with authorisation and access control) which could be implemented either by using the existing OMG Security Service (at level one or above), or by a GL Facility implementation itself if no OMG Security Service implementation is available.

C.2.2 OMG Object Transaction Service (OTS)

While this specification makes no explicit use of the OMG Object Transaction Service itself, it is likely that implementations targeted at large enterprises will take advantage of the facilities of this service for scalability. The specification neither precludes nor mandates use of the OTS.

C.2.3 OMG Unified Modelling Language (UML)

The underlying models derived by the co-submitters as part of their work on the design of the interface structure are based on the RM-ODP approach and documented using OMG UML. Additional supporting documentation in the form of ODP based Enterprise and Information viewpoint specifications have been provided separately in OMG DTC documents finance/98-12-01 and finance/98-12-02 respectively.

C.2.4 OMG Currency Facility

This specification uses the OMG `FbcCurrency` Currency Facility, which provides a `Money` type used in this specification. As some specifications and features are not yet finalised by OMG nor available in actual implementations, the submitters have compiled the GL Facility IDL by changing *value* types defined in the Currency Facility to *interface* types.

C.2.5 OMG Relationship Service

Although this specification does not use this service itself, it is used by the Currency Facility (see “Currency Facility” above).

C.2.6 OMG Query Service

Although this specification does not use this service itself, it is used by the Currency Facility (see “Currency Facility” above).

C.3 Relationship to CORBA

The General Ledger Facility assumes the use of a CORBA compliant ORB.

C.4 Relationship to the OMG Object Model

The General Ledger Facility conforms to the OMG Object Model.