Initial Submission
OMG's Finance DTF RFP 4

General Ledger Facility

An initial, combined submission from:

Stanford Software International Ltd and the OMG Domain Contributing Members of the European Union's COMPASS Project; Economica AS and Real Objects Ltd.

OMG General Ledger Facility

Revision 1.1

July 3rd, 1998

OMG DTC Document finance/98-07-02

Acknowledgements

The submitters wish to formally acknowledge and express their gratitude to the European Commission for its support. Without the support of the European Commission, this project would not have been possible. The following individuals and organisations provided significant contributions to the combined submission: Jack Hassall, John Eaton, Gary Gray and Mike Wilcock of Stanford Software International; Eric Leach, ELM Ltd; Tom Mowbray, Blueprint Technologies Inc; Arne-Jorgen Berre, Tor Neple and Anton Landmark of SINTEF; Roger Barnett, Adrian St. John, Real Objects Ltd; Morten Jacobsen, Amund Aarsten and Ole Petter Ovre of Economica and Colin Mason and Keith Thomson of The Software Box Ltd.

Special Thanks

The following list of individuals is extensive and is naturally incomplete, however, in varying degrees of proportionality, special thanks must go to the following individuals from all over the world without whose support and encouragement over the years, this submission would not have been possible: Alan O' Callaghan, Andrew Rosamond, Andrew Watson, Anthony Kaiser, Beth Grossman, Dr. Bhavani Thuraisingham, Bill Brown, Bill Cox, Bill Hatch, Bill Heartha, Bill Hoffman, Bryan Wood, Cheryl Bisonette, Cappie Jarboe, Carol Tyson, Cathy Batzer, Cheryl Rocheleau, Chris Sluman, Chris Stone, Chuck Alvarez, Chuck Lockard, Cory Casanave, Dai Clegg, Dan Franz, Dave Curtis, Dave Jack, Dave Zenie, David Newman, David Guest, Don Shepard, Doug Moss, Ed Cobb, Eric Castain, Erlund Stav, Fred Cummins, Fred Waskiewicz, Gary Word, Gene Jarboe, Geoff Cave, Gillian Lefel, Graham Lea, Haim Kilov, Henry Rothkopf, Hilary Khan, Hiroki Kamata, Huet Landry, Iain Houston, Ian Corden, Ian Hugo, Jean-Marie Chauvet, Jim Rye, Joe DiLiberto, John Wieler, Jon Siegel, Junishi Suzuki, Jurgen Boldt, Karen Oulton, Karen Wastling, Kate Brown, Kate Mowbray CPA, Keith Oulton, Ken Kolence, Ken McCoy, Kevin Schipani, Kevin Tyson, Kurt Ramin, Larry Gray, Larry Johnson, Lydia Bennett, Mark Holtom, Mark Ryland, Martin Chapman, Martin Fowler, Melony Katz, Michael Guttman, Nick Langley, Norm Eko, Pat Elizondo, Pat Mallet, Peter Marshall, Polar Humen, Raphael Malveau, Rich Limieux, Richard Carr, Richard Lowrie, Dr. Richard Soley, Robert Mickley, Roberto Zicari, Rod Newing, Ron Zahavi, Ross Mayne, Roz Sluman, Rudi Reiss, Samit Khosla, Shel Sutton, Steve Macnamara, Steve van Noort, Steve Turner, Steve Vinowski, Steve Wolfe, Sumit Ray, Thelma Leach, Tom Kilburn, Tom Rutt, Dr. Tom Mowbray, Tony Ackroyd, Tony Caballero, Toshiaki Kurokawa, Vicki Chadwick and Yllona Richardson.

The organisations represented by the above list of individuals include: ACORD, Apptest, AT Kearney, AT&T Lucent, Barclays Bank, BEA Systems, Blackwatch Technologies, Broadcom, Canadian Imperial Bank of Commerce, Citicorp, Concept5 Technologies, Cyborg Systems, Data Access Technologies, De Montfort University, DEC, US Defence Information Systems Agency Center for Standards, DNS Technologies, Ecsoft, EDS, ELM, Enterprise Engineering Associates, Fannie-Mae, Genesis Development Corporation, Heterodox, IBM, ICL Object Software Laboratories, International Accounting Standards Committee, Iona Technologies, J.P. Morgan, Kolence Associates, Level-7, LogON Technology, Macqaurie Bank, Merrill Lynch, Microsoft, MITRE, MSC, US National Industrial Information Infrastructure Protocols (NIIIP) Consortium, US National Security Agency, Neuron Data, Novell, Object Management Group,

Open-IT, Oracle, Protocol Systems, Real Objects, Restore Computer Services, Sematech, SINTEF, Soken Planning, Tandem, Technium, The Objective Technology Group, Traveller's Group, University of Manchester, University of Sheffield and Wells Fargo Bank.

Table Of Contents

Section I: General Ledger (GL) Facility Overview

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 organisations and individual proprietorships. 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 standardisation. Additionally, many other accounting applications including Accounts Payable, Accounts Receivable, Inventory, Sales, 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.

"Monetary calculation is the guiding star of action under the social system of division of labour. 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 civilisation 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.

GL Facility Structure

The General Ledger (GL) Facility specifies interfaces that encapsulate distributed object frameworks implementing Accounting General Ledgers, these GL's 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 a complete set of GL services that fully support the implementation of accounting clients that need to interoperate with one or more GL 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 and the Esprit COMPASS project.

GL Interface Summary

The General Ledger Facility defines interfaces (using OMG/ISO IDL) to support the capabilities as mentioned previously. The following table gives a high level description of the General Ledger Facility interfaces. Subsequent sections describe the interfaces in more detail.

Interface	*Purpose*	*Primary Client(s)*
GLProfile	Client Session Establishment	All GL clients
GLBookKeeping	Data entry	Data entry clients
GLRetrieval	Data extraction	Reporting clients
GLAccountLifecycle	Account lifecycle management	GL administration clients
GLIntegrity	Data integrity checks	GL administration clients
GLFacilityLifecycle	GL lifecycle management	GL administration clients

Table 1-1 Synopsis of General Ledger Facility Interfaces

Service Level Interfaces

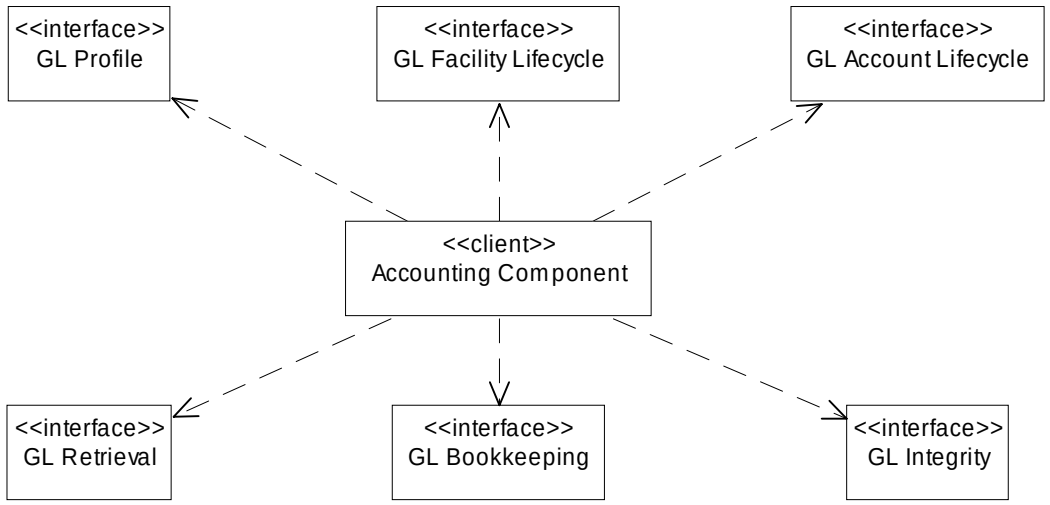

Figure 1-1: Illustrates the different service level interfaces that comprise the General Ledger Facility, as documented in Table 1-1.

Points of Contact

All questions about this submission should be directed to:

Jack Hassall
Stanford Software International Ltd
Address: The Hollygate, Chestergate
Stockport
SK3 0BD
United Kingdom
E-mail: jack_hassall@omg.org
Tel.: +44 161 480 4051
Fax: +44 161 429 0966

Other contacts include:

Roger Barnett
Real Objects Ltd
Address: 118-120 Warwick Street
Leamington Spa
CV32 4QY
United Kingdom
E-mail: roger@realobj.co.uk
Tel.: +44 1926 332116 ex 239
Fax: +44 1926 883370

Arne-Jørgen Berre
SINTEF Telecom and Informatics
Address: Forskningsveien 1
Po Box: 124, Blindern, N-0314
Oslo, Norway
E-mail: arne.j.berre@informatics.sintef.no
Tel.: +47 2206 7452
Fax: +47 2206 7350

Morten Jacobsen
Economica AS
Address: Olaf Helsetsvei 6
Po Box: 70 Bogerud, N-0621
Oslo, Norway
E-mail: morten@economica.com
Tel.: +47 920 97 818
Fax: +47 6258 0197

Section II: Module FdGeneralLedger

The IDL code in this specification follows the established conventions used by many of the previously adopted OMG IDL specifications. In the following sections, IDL code is set in courier font. Specification semantics are set in Times New Roman font.

The FdGeneralLedger module defines 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 client does not have to depend upon interfaces it doesn't use.

Included OMG/ISO IDL Files

```
#include <FdCurrency.idl>
```

The GL Facility uses currency types defined from an external currency specification (See Appendix A for Service Dependencies).

Module FdGeneralLedger

```
module FdGeneralLedger {
```

The module statement establishes the syntactic scope for the GL Facility definitions. The module name uses the financial domain naming standard with the "Fd" prefix.

GL Facility Invariants

These are the key assumptions regarding the responsibilities of GL Facility implementations.

- *The GL Facility maintains state for each client session. For example, each client session concerns only one known company and the company's established chart of accounts.*
- *There is a one-to-one mapping between each company 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 client session are constrained by session-specific GL policies. See GLFacilityLifecycle for administrative operations.*

GL Facility Environment Contract

These are the key assumptions provisioned for the environmental objects containing and managing the GL Facility.

- *The GL Facility assumes that client authentication for the security policy domain has occurred prior to access to GL interfaces. See Security Service Dependencies, Appendix A.*

- *The GL Facility assumes that access controls will be applied according to system domain policies during prior to and during client sessions. For example, the passing of clear-text parameters in operation invocations will be protected from unauthorised access or disclosure.*

- *The only interface provided to GL clients prior to GL session establishment is the GLProfile interface.*

- *The environment shall not disclose other GL interfaces to GL clients. That is the responsibility of the GLProfile interface. For example, only the GLProfile interface shall be advertised in the Trader Service and Name Service. Other GL interfaces are provided by the GLProfile interface, subsequent to GL client session establishment.*

GL Forward Declarations

```
interface GLProfile;                    // establish client session
interface GLBookKeeping;                // information entry
interface GLRetrieval;                  // information acquisition
interface GLIntegrity;                  // information integrity checks
interface GLAccountLifecycle;           // GL Account lifecycle management
interface GLFacilityLifecycle;          // GL Facility lifecycle management
```

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

GL Typedefs

The FdGeneralLedger module defines several types for the 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.

GL Basic Types

```
typedef sequence<boolean> booleanList;
typedef sequence<wstring> wstringList;
```

Collection types for booleans and wide strings. All the strings used in the module are wstrings.

```
struct NameValue {
    wstring name;
    wstring value; };                               // TBD.
typedef sequence<NameValue> NameValueList;
```

Name-value pair and collection of name-value pairs.

```
typedef FdCurrency::Date Date;
typedef FdCurrency::Money Money;
typedef wstring Currency;                           // ISO currency mnemonic
```

Types imported from an external currency specification, module FdCurrency. These types are used opaquely in this specification.

GL Account Information

```
enum ChartKind {DEFAULT_NOMINAL, EXISTING_CHART, EMPTY_LEDGER };
```

Defines the different kinds of Charts of Accounts schemas for the purposes of initialisation. Used when setting up the chart of accounts for a company's ledger.

```
struct AccountInfo {
    wstring acc_ref;
    wstring description; };
typedef sequence<AccountInfo> AccountInfoList;

enum AccountKind { CASH, BANK, CONTROL, REGULAR };
```

Identifies the pre-defined types of GL accounts. A GL account is a "regular" GL account by default. The GL maintains additional state values for GL accounts. Cash and Bank accounts are special designations of GL accounts, that otherwise behave like "regular" GL accounts.

Summary information about GL accounts, with their identifier and descriptive name.

```
struct Account {
        wstring         GLAcc_ref;          // GL Account reference
        wstring         GLAcc_name;         // name
        wstring         GLreporting_code;   // grouping code
        Currency        default_currency;
        Money           balance;
        boolean         is_control;
        Money           mth_bal;
        Money           ytd_bal;
        wstring         con_acc_kind;
        wstring         con_acc_desc;
        wstringList     optional_fields; };
```

The account structure is a description of an account in the General Ledger. The field `GLAcc_ref` is a unique reference to the account within the GL Facility. The field `GLAcc_name` is a descriptive client specified name. The `GLreporting_code` is a reporting code that may be used as a synonym for the account name. A default_currency is specified for each account.. GL accounts and control accounts are differentiated by the `is_control` boolean. The fields `balance`, `mth_bal` (monthly), and `ytd_bal` (year to date) are state values maintained for GL control accounts. The fields `con_acc_kind` and `con_acc_desc` denote characteristics of control accounts. The `optional_fields` contain implementation specific extensions.

```
typedef sequence<Account> AccountList;
```

A collection of GL account descriptions.

GL Periods and Transaction Information

```
enum PeriodKind { NO_DATE, WEEK, MONTH, QUARTER, YEAR };          // TBD.
```

An accounting period can be undated (NO_DATE), weekly, monthly, quarterly, or yearly.

```
struct AccountingPeriod {                                         // TBD.
    wstring period_name;
    PeriodKind period_kind;
    Date start_date;
    Date end_date; };
```

Transactions posted to the GL belong to accounting periods, which can correspond to a date range or be undated. Undated periods are usually used for special transactions connected to year end closing procedures. Accounting periods have a client defined period name. Accounting periods also have a start date and an end date which are an inclusive in the specified period.

```
struct HistorySpec{
        wstring lower_acc_ref, upper_acc_ref;
        wstring start_period, end_period;                         // TBD.
        Date start_date, end_date;
        wstring lower_trans_no, upper_trans_no; };
typedef sequence<HistorySpec> HistorySpecList;
```

Used for retrieving a subset of the transactions in the ledger.

```
struct ControlAccInfo {
    wstringList     control_acc_names;
    wstringList     control_acc_ref_nos;
    unsigned short  max_bank_accs;                                // TBD.
    wstringList     bank_acc_ref_nos; };
typedef sequence <ControlAccInfo> ControlAccInfoList;
```

Used for retrieving information about GL control accounts.

```
struct TransactionInfo {                                          // TBD.
    wstring         trans_no;
    wstring         trans_kind;
    wstring         period_id;              // not in appendix b
    Date            trans_date; };
```

Summary information of a Transaction.

```
typedefsequence <TransactionInfo> TransactionInfoList;
```

Summary information about a list of GL transactions posted to the GL

```
struct Entry {                                                    // TBD.
    unsigned long   trans_no;
    Date            entered_date;
    wstring         account_no;
    wstringList     dimension_accounts;
    Money           amount;
    DDecimal        quantity;
    wstring         description;
    wstring         rule_ref;
    wstring         invoice_no;
    wstring         document_ref;
    wstring         user_name;
    ValueList       optional_fields;
};
```

A column entry in the ledger.

```
typedef sequence<Entry> EntryList;
```

A list of column entries.

```
struct Transaction {
    unsigned long   trans_no;
    wstring         trans_kind;
    wstring         period_id;
    Date            trans_date;
    wstring         document_ref;
    EntryList       entries;
    ValueList       optional_fields;
};
```

A row in the ledger, i.e. a balanced transaction. The sum of the entries' amounts must be 0.

```
typedef sequence<Transaction> TransactionList
```

A list of transactions. // TBD.

GL Data Type Argument Exceptions

The type booleanList is used to indicate the position of the error. For example, if the error is in a struct type, the booleanList indicates the bad struct member, positional order, starting with 0. For Accounts, Transactions, and Entries, the position is indexed by the GLParameterIds.

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

        exception BadChartKind { wstring error
            ChartKind bad_value; };

        exception BadSelection { wstring error;
            unsigned long selection_code;
            booleanList bad_members; };

        exception BadTransaction { wstring error;
             wstring trans_no; booleanList bad_fields; };

        exception BadAccountKind {wstring error;
            AccountKind bad_value; };

        exception BadHistorySpec { wstring error;
            booleanList bad_members; };

    exception BadPeriod {
           wstring error;
           wstring period_id; };                        // TBD.
```

Other Exceptions

```
        exception BadName { wstring error, bad_value; };

        exception BadAccountRef { wstring error;
           wstring bad_value; };
```

GL account reference does not exist, or, when creating a new account, the account reference is invalid or, there is already a GL account with that reference.

```
        exception BadTransNo { wstring error;
           wstring bad_value; };

        exception NoChartOfAccounts { wstring error; };
```

There is no chart of accounts. The NoChartOfAccounts exception is used explicitly on GL initialisation operations. Other exceptions may be raised if the Chart of Accounts is uninitialized or otherwise improperly configured.

```
exception CannotRemove ( wstring error; };
```

It is not possible to delete or remove this object.

```
exception ProfileError { wstring error; };              // TBD.

exception UnknownCompany( wstring error, bad_value; );
```

Company name does not match a known chart of accounts. bad_value is the erroneous company name.

```
exception MaxExceeded { wstring error;
      unsigned long max_amount; };
```

An implementation-specific amount was exceeded.

```
exception BadIntegrity { wstring error;
      any bad_value; };
```

Indicates a failure of a GL integrity check.

```
exception BadAccountName { wstring error;
      wstring bad_value; };
```

Account name is invalid.

```
exception BadReportingCode { wstring error;
      wstring bad_value; };
```

Reporting code is invalid.

Section III: GL Profile Interface

The GLProfile is the initial interface used to establish a client session. A client session must be established prior to use of the General Ledger Facility. Each client session must use a unique instance of GLProfile

GLProfile Operation: get_default_company_name

```
wstring get_default_company_name()
    raises ( NoChartOfAccounts );
```

Description
Each GL Facility can manage the General Ledgers of many different companies (one GL per company.) One of the companies is designated as the default company; its name can be retrieved with this operation.

Precondition
none

Input parameters
none

Output parameters
none

Return value
Returns the default company name as a wstring.

Exceptions
NoChartOfAccounts: Raised when there are no ledgers (and hence no default company name) available.

Postcondition
none

GLProfile Operation: get_GL_company_names

```
wstringList get_GL_company_names()
    raises ( NoChartOfAccounts );
```

Description
Allows the clients to retrieve the company names of the available ledgers.

Precondition
This operation can be called prior to GL login.

Input Parameters
none

Output Parameters
none

Return Value
Returns wstringList with a sequence of all company names available in the General Ledger Facility.

Exceptions
NoChartOfAccounts: Raised when there are no ledgers available.

Postcondition
none

GLProfile Operation: get_current_system_date

```
Date get_current_system_date();
```

Description
Allows the clients to retrieve the current date used by the GL Facility

Precondition
none

Input Parameters
none

Output Parameters
none

Return Value
Returns a `Date` object.

Exceptions
none

Postcondition
none

GLProfile Operation: general_ledger_open

```
wstring general_ledger_open (
            in wstring company_name, in Date system_date )
            raises ( UnknownCompany, ProfileError, BadDate );
```

Description
Establishes client session for a General Ledger by company name. `system_date` is the effective date of the session. This is so any updates to the General Ledger can appear in the correct period.

Precondition
none

Input Parameters
wstring company_name: The company name of the ledger to select.
Date system_date: The effective system date for this session.

Output Parameters
none

Return Value
Returns a ClientProfile struct with information about the client session..

Exceptions
UnknownCompany: raised if a ledger for the company named company_name is not known to the General Ledger Facility.

Postcondition
A client session is established against the ledger identified by company_name. If the client is already logged in, an exception is raised.

GLProfile Operation: book_keeping

```
GLBookKeeping book_keeping();
```

Description
This method retrieves a reference to the GLBookKeeping interface for the current company.

Precondition
A client session must have been established with general_ledger_open.

Input Parameters
none

Output Parameters
none

Return Value
Returns a GLBookKeeping for use in the current session. Once the session has ended the returned GLBookKeeping is no longer a valid object.

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call.

Postcondition
None

GLProfile Operation: retrieval

```
GLRetrieval retrieval()raises (ProfileError);
```

Description
This method retrieves a reference to the GLRetrieval interface for the current company.

Precondition
A client session must have been established with general_ledger_open.

Input Parameters
none

Output Parameters
none

Return Value
Returns a GLRetrieval for use in the current session. Once the session has ended the returned GLRetrieval is no longer a valid object.

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call.

Postcondition
None

GLProfile Operation: integrity

```
GLIntegrity integrity() raises (ProfileError);
```

Description
This method retrieves a reference to the GLIntegrity interface for the current company.

Precondition
A client session must have been established with general_ledger_open.

Input Parameters
none

Output Parameters
none

Return Value
Returns a GLIntegrity for use in the current session. Once the session has ended the returned GLIntegrity is no longer a valid object.

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call.

Postcondition
None

GLProfile Operation: account_lifecycle

```
GLAccountLifecycle account_lifecycle()raises (ProfileError);
```

Description
This method retrieves a reference to the GLAccountLifecycle interface for the current company.

Precondition
A client session must have been established with general_ledger_open.

Input Parameters
none

Output Parameters
none

Return Value
Returns a GLAccountLifecycle for use in the current session. Once the session has ended the returned GLAccountLifecycle is no longer a valid object.

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call.

Postcondition
None

GLProfile Operation: Facility_lifecycle

```
GLFacilityLifecycle Facility_lifecycle()raises (ProfileError);
```

Description
This method retrieves a reference to the GLFacilityLifecycle interface for the current company.

Precondition
A client session must have been established with general_ledger_open.

Input Parameters
none

Output Parameters
none

Return Value
Returns a GLFacilityLifecycle for use in the current session. Once the session has ended the returned GLFacilityLifecycle is no longer a valid object..

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call.

Postcondition
None

GLProfile Operation: get_client_company_name

```
wstring get_client_company_name() raises (ProfileError, NoChartOfAccounts);
```

Description
This method retrieves the name of the currently selected company.

Precondition
A client session must have been established with general_ledger_open.

Input Parameters
none

Output Parameters
none

Return Value
Returns the currently active company name.

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call. NoChartOfAccounts is raised when there are no companies in the General Ledger.

Postcondition
None

GLProfile Operation: get_current_period

```
AccountingPeriod get_current_period() raises (ProfileError);
```

Description
Each ledger implementation maintains a default period, which is typically the last open period. The get_current_period() operation retrieves this default period for the current company.

Precondition
A client session must have been established with general_ledger_open.

Input Parameters
none

Output Parameters
none

Return Value
Returns an AccountingPeriod struct with information about the current period for the current ledger.

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call.

Postcondition
none

GLProfile Operation: get_GLReport_codes

```
wstringList get_GLReport_codes()raises (ProfileError);
```

Description
Each ledger uses a set of reporting codes, which are typically used to group accounts in reports. This operation retrieves the reporting codes available for the current client.

Precondition
A client session must have been established with general_ledger_open. A set of reporting codes should exist for the ledger. For implementations which do not supply a default set of reporting codes, the client can ensure this by calling the GLFacilityLifecycle operation set_GLReport_codes().

Input Parameters
none

Output Parameters
none

Return Value
Returns a wstringList with a list of report codes valid for the current session.

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call.

GLProfile Operation: get_default_currency

```
wstring get_default_currency()raises (ProfileError);
```

Description
Each ledger maintains a default currency, which is used when creating new accounts (see the createAccount operation in GLAccountLifecycle.)

Precondition
A client session must have been established with general_ledger_open.

Input Parameters
none

Output Parameters
none

Return Value
Returns a wstring containing the default currency mnemonic for the current session.

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call.

Postcondition
none

GLProfile Operation: get_known_currencies

```
wstringList get_known_currencies() raises (ProfileError);
```

Description
The Money values in financial transactions posted to the GL must have a valid currency mnemonic and Money attribute. This method allows the client to retrieve all available currencies

Precondition
A client session must have been established with general_ledger_open.

Input Parameters
none

Output Parameters
none

Return Value
Returns wstringList with a list of known currency mnemonics valid for the current session.

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call.

Postcondition
none

GLProfile Operation: get_dimension_names

```
wstringList get_dimension_names() raises (ProfileError);
```

Description
Dimensions are orthogonal axes in proportion to an account. get_dimension_names finds the names that match the clients account.

Precondition
none

Input Parameters
none

Output Parameters
none

Return Value
Returns wstringList:with the names of the available dimensions.

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call.

Postcondition
none // TBD.

GLProfile Operation: quit

```
void quit() raises (ProfileError);
```

Description
Closes down the current client session.

Precondition
A client session must have been established with general_ledger_open.

Input Parameters
none

Output Parameters
none

Return Value
none

Exceptions
ProfileError: The client has not successfully established a session with general_ledger_open prior to the call.

Postcondition
Once this method is invoked, all references to other interfaces in the General Ledger immediately become invalid. This method also invalidates any references to the current GLProfile so that the resource is properly released. However, compliant implementations need not enforce this.

Section IV: GLBookKeeping Interface

The GLBookkeeping interface is used for entering new transactions in the ledger of the company selected at login.

GLBookKeeping Operation: post

```
void post(in Transaction single_transaction)
    raises (BadTransaction);
```

Description
Posts a single transaction to the ledger.

Precondition
none

Input Parameters
transaction single_transaction: The transaction to post.

Output Parameters
none

Return Value
none

Exceptions
BadTransaction: One of the fields in trans or its entries have an illegal value, or the transaction is not balanced (the sum of its entries is not 0.)

Postcondition
The transaction is added to the ledger, and the balances of the accounts referenced by the entries are updated.

GLBookKeeping Operation: post_batch

```
void post_batch ( in TransactionList transactions ) raises ( BadTransaction );
```

Description
Posts a list of transactions to the ledger.

Precondition
none

Input Parameters
TransactionList transactions: The list containing the transactions to post.

Output Parameters
none

Return Value
none

Exceptions
BadTransaction: One of the fields in trans or its entries have an illegal value, or the transaction is not balanced (the sum of its entries is not 0.) The transaction causing the error is identified with the trans_no variable of the exception.

Postcondition
The ledger contains the transactions in the transactions list. If one of the transactions causes an exception to be raised, none of the transactions in the list are written to the ledger. The balances of the accounts referenced by the entries in the transactions are updated.

Section V: GLRetrieval Interface

The GLRetrieval interface supports client reporting functions for the chart of accounts and the transactions in the ledger for the current company.

Chart of Accounts Information

The following operations provide information about the chart of accounts in the current company's ledger.

GLRetrieval Operation: number_of_accounts

```
unsigned long number_of_accounts();
```

Description
Retrieves the number of accounts in the ledger for the current company.

Precondition
none

Input Parameters
none

Output Parameters
none

Return value
Returns the number of accounts as an unsigned long.

Exceptions
none

Postcondition
none

GLRetrieval Operation: get_account_info

```
AccountInfoList get_account_info(
    in AccountKind type_of_account)
    raises (BadAccountKind);
```

Description
Allows the client to retrieve summary information about the accounts of a specific type.

Precondition
none

Input Parameters
AccountKind type_of_account: The account type for which the summary account information are to be retrieved.

Output Parameters
none

Return Value
Returns an AccountInfoList with summary information for all the accounts of the specified type.

Exceptions
BadAccountKind: The parameter type_of_account is not a valid AccountKind.

Postcondition
none

GLRetrieval Operation: get_all_account_info

```
AccountInfoList get_all_account_info();
```

Description
Retrieves summary information for all the accounts in the ledger.

Precondition
none

Input Parameters
none

Output Parameters
none

Return Value
Returns an AccountInfoList with summary information for all the accounts in the ledger.

Exceptions
none

Postcondition
none

GLRetrieval Operation: get_account

```
Account get_account(in wstring GLAcc_ref)
    raises (BadAccountRef);
```

Description
Retrieves a single account from the current ledger.

Precondition
none

Input Parameters
wstring GLAcc_ref: The number of the account to be retrieved.

Output Parameters
none

Return Value
Returns an Account struct with a full description of the account identified by GLAcc_ref.

Exceptions
BadAccountRef: The account reference does not exist in the ledger.

Postcondition
none

GLRetrieval Operation: get_multiple_accounts

```
AccountList get_multiple_accounts(
    in wstringList GLAcc_refs)
    raises (BadAccountRef);
```

Description
Retrieves a set of accounts from the ledger.

Precondition
none

Input Parameters
wstringList GLAcc_refs: List containing the numbers of the accounts to be retrieved.

Output Parameters
none

Return Value
Returns an AccountList containing full descriptions for the accounts identified by the account references in GLAcc_refs

Exceptions
BadAccountRef: An account reference in the GLAcc_refs list parameter does not exist in the ledger.

Postcondition
none

GLRetrieval Operation: get_accounts_from_GLreporting_code

```
AccountList get_accounts_from_GLreporting_code(
    in wstring GLreporting_code)
    raises (BadReportingCode);
```

Description
Allows the client to retrieve accounts for a certain reporting code.

Precondition
none

Input Parameters
wstring GLreporting_code: The reporting code for which accounts are to be retrieved.

Output Parameters
none

Return Value
Returns an AccountList with the accounts having the given reporting code.

Exceptions
BadReportingCode: The reporting code is not valid.

Postcondition
none // TBD.

GLRetrieval Operation: get_control_acc_info

```
AccountInfoList get_control_acc_info();
```

Description
Allows the client to retrieve the control accounts in the ledger.

Precondition
none

Input Parameters
none

Output Parameters
none

Return Value
Returns an AccountInfoList containing summary information for all the control accounts in the ledger.

Exceptions
none

Postcondition
none // TBD.

General Ledger Transactions Retrieval

These operations provide information about transactions which have been entered in the current company's ledger.

GLRetrieval Operation: number_of_current_transactions

```
unsigned long number_of_current_transactions();
```

Description
Retrieves the total number of financial transactions posted to the current company's General Ledger.

Precondition
none

Input Parameters
none

Output Parameters
none

Return Value
Returns the number of transactions as an unsigned long.

Exceptions
none

Postcondition
none

GLRetrieval Operation: get_transaction_info

```
TransactionInfoList get_transaction_info(in trans_no) raises (BadTransNo);
```

Description
Retrieves summary information for all specified financial transactions posted to General Ledger for a given GL account reference.

Precondition
none

Input Parameters
trans_no // TBD.

Output Parameters
none

Return Value
Returns a TransactionInfoList containing summary information for all transactions related to the specified GL account reference.

Exceptions
BadTransNo is raised if the specified GL transaction reference is invalid.

Postcondition
none

GLRetrieval Operation: get_multiple_transaction_info

```
TransactionInfoList get_multiple_transaction_info ( in HistorySpec
history_range )raises (BadHistorySpec, MaxExceeded );
```

Description
Retrieves summary information for all the transaction in the current company's ledger as specified by the HistorySpec.

Precondition
none

Input Parameters
HistorySpec

Output Parameters
none

Return Value
Returns a TransactionInfoList containing summary information for all the ledger transactions.

Exceptions
none

Postcondition
none

GLRetrieval Operation: get_current_history_range

```
HistorySpec get_current_history_range( out unsigned long
number_of_transactions);
```

Description
Subsets of the transactions in the ledger are specified with the HistorySpec struct. This operation retrieves the largest range of filter values for the transactions in the current company's ledger, such as the first and last accounting period and the range of transaction numbers.

Precondition
none

Input Parameters
none

Output Parameters
unsigned long number_of_transactions: The number of transactions that will fit the returned history spec, i.e. the total number of transactions in the ledger.

Return Value
Returns a HistorySpec with the maximum range for all the history dimensions of the transactions in the ledger.

Exceptions
none

Postcondition
none

GLRetrieval Operation: number_of_history_transactions

```
unsigned long number_of_history_transactions(
    in HistorySpec history_range)
    raises (BadHistorySpec);
```

Description
For a given account history specification range, this operation returns the number of transactions in the chart of accounts that match the history specification.

Precondition
none

Input Parameters
HistorySpec history_range: The transaction filter for which the transaction count is desired.

Output Parameters
none

Return Value
Returns the number of transactions which fall within the range specified by history_range as an unsigned long.

Exceptions
BadHistorySpec: One or more fields of the history_range parameter contains illegal values.

Postcondition
none

GLRetrieval Operation: get_multiple_transactions

```
TransactionList get_multiple_transactions(
    in HistorySpec history_range)
    raises (BadHistorySpec, MaxExceeded);
```

Description
Retrieves a subset of the transactions in the current company's ledger.

Precondition
none

Input Parameters
HistorySpec history_range: The filter to use for selecting transactions from the ledger.

Output Parameters
none

Return Value
Returns a TransactionList containing the transactions in the ledger, which fall into the range history_range.

Exceptions
BadHistorySpec: one or more fields of the history_range parameter contains illegal values
MaxExceeded: an implementation specific maximum value has been exceeded.

Postcondition
none

GLRetrieval Operation: get_current_transactions

```
TransactionList get_current_transactions();
```

Description
Retrieves all the transactions in the ledger. This may return a very large number of transactions. Most client applications will use the get_multiple_transactions operations to retrieve a subset of the transactions at a time.

Precondition
none

Input Parameters
none

Output Parameters
none

Return Value
Returns a TransactionList containing all the transactions in the ledger.

Exceptions
none

Postcondition
none

Section VI: GLAccountLifecycle Interface

The GLAccountLifecycle service manages the lifecycle of the accounts in the ledger, facilitating the customisation of the chart of account selected when the ledger was created. Only client sessions with GL Manager privileges can use the operations in the interface.

GLAccountLifecycle Operation: createAccount

```
void createAccount(in wstring GLAcc_ref,
    in wstring GLAcc_name,
    in boolean is_nominal_account,
    in boolean is_control_account,                    // TBD.
    in wstring reporting_code)
    raises (BadAccountRef, BadAccountName,
    BadReportingCode);
```

Description
Creates a new account, identified by GLAcc_ref.

Precondition
The client session must have been established with Manager privileges.

Input Parameters
wstring GLAcc_ref: unique identifier for accounts. wstring GLAcc_name: descriptive name for the account. Whether an account name has to be unique or not is implementation-defined. Boolean is_nominal_account: true if the account is nominal (its balance is reset at the end of the accounting year.) Boolean is_control_account: true if the account is a control account. wstring reporting_code: the reporting code, which must be one of the codes passed to the GLFacilityLifecycle operation set_GLReport_codes().

Output Parameters
none

Return Value
none

Exceptions
BadAccountRef: raised if GLAcc_ref is not unique, empty, or otherwise unacceptable to the underlying implementation. BadAccountName: raised if GLAcc_name is empty, or otherwise unacceptable to the underlying implementation. BadReportingCode: raised if reporting_code is not one of the allowed values..

Postcondition
The new account is added to the ledger. The default currency of the new account is set to the default currency of the ledger.

GLAccountLifecycle Operation: removeAccount

```
void removeAccount(in wstring GLAcc_ref)
    raises (BadAccountRef,
    CannotRemove);
```

Description
Removes the account identified by GLAcc_ref from the chart of accounts.

Precondition
The client session must have been established with Manager privileges. The account cannot be in use; accounts which are in use cannot be deleted. An account is in use when there are associated financial transactions in the ledger, or it is a control account referring to a non-empty set of other accounts, or the balance of the account is non-zero.

Input Parameters
wstring GLAcc_ref: unique identifier for the account.

Output Parameters
none

Return Value
none

Exceptions
BadAccountRef: raised if the account reference GLAcc_ref is not present in the ledger.
CannotRemove: raised if the account is in use.

Postcondition
none

GLAccountLifecycle Operation: modify_account

```
void modify_account(in wstring GLAcc_ref,
    in wstring new_GLAcc_name,
    in wstring new_reporting_code)
    raises (BadAccountRef,
    BadAccountName,
    BadReportingCode);
```

Description
Modifies the descriptive name and/or the reporting code associated with the account identified by GLAcc_ref.

Precondition
The client session must have been established with Manager privileges.

Input Parameters
wstring GLAcc_ref: identifies the account to be modified.
wstring new_GLAcc_name: new descriptive name for the account.
wstring new_reporting_code: new reporting code for the account.

Output Parameters
none

Return Value
none

Exceptions
BadAccountRef: raised if the GLAcc_ref not is identified or another account has the same id.
BadAccountName: raised if the GLAcc_name is empty or otherwise unacceptable to the underlying implementation. BadReportingCode: The reporting code is invalid.

Postcondition
none

GLAccountLifecycle Operation: close_accounting_period

```
void close_accounting_period(
   in wstring period_id)
   raises (BadPeriod);
```

Description
Closes the accounting period identified by period_id. No more transactions can be posted to the ledger for a closed period.

Precondition
The client session must have been established with Manager privileges.
The period must be open. A period in an accounting year cannot be closed if the previous accounting year has not been closed with close_accounting_year. All preceding accounting periods must be closed.

Input Parameters
wstring period_id: indicate the account period to be closed.

Output Parameters
none

Return Value
none.

Exceptions
BadPeriod: raised if period_id does not exist or the period is already closed.

Postcondition
none // TBD.

GLAccountLifecycle Operation: close_accounting_year

```
void close_accounting_year(
    in wstring last_period_in_year)
    raises(BadPeriod);
```

Description
Marks the accounting year whose last period is last_period_in_year as closed. Note that year-end closing can be a complex process, and it varies greatly between different implementations and even users. Therefore, this operation does not perform any specific year-end processing, such as transferring the balances of nominal (e.g. profit and loss) accounts to the appropriate balance sheet accounts. Such operations are intended to be performed by components or applications using any necessary GL interfaces for the purpose.

Precondition
The client session must have been established with Manager privileges.
The period last_period_in_year must be closed.

Input Parameters
wstring last_period_in_year: Identifies the last period in the year to be closed.

Output Parameters
none

Return Value
none

Exceptions
BadPeriod: The period does not exist, is not closed, or is not the last period in an accounting year.

Postcondition
none // TBD.

Section VII: GLIntegrity Interface

The GLIntegrity interface provides integrity checks of the chart of accounts and transactions in the ledger of the current company.

GLIntegrity Operation: get_dynamic_selection

```
wstringList get_dynamic_selection();
```

Description
Returns a list of available integrity tests. Each integrity test is identified by a name.

Precondition
none

Input Parameters
none

Output Parameters
none

Return Value
Returns a wstringList containing the implementation-defined list of integrity checks.

Exceptions
none

Postcondition
none

GLIntegrity Operation: check_integrity

```
boolean check_integrity(
    in wstring integrity_check_selection)
    raises (BadSelection, BadIntegrity);
```

Description
Performs the indicated integrity check.

Precondition
The integrity check must be one of those returned by get_dynamic_selection, which therefore has to be called first.

Input Parameters
wstring integrity_check_selection: Identifies the integrity check to perform.

Output Parameters
none

Return Value
Returns true if the integrity test passes, false if there are warnings.

Exceptions
BadSelection: The selected test is not one of the available integrity tests. BadIntegrity: The integrity test failed.

Postcondition
Implementation-defined.

Section VIII: GLFacilityLifecycle Interface

The GLFacilityLifecycle operations are used to manipulate the information in the Facility which is independent of the individual ledgers. This information include users and their access rights, companies and their chart of accounts, and other information which define the allowed values for some of the fields of the Account and Transaction structs.

GLFacilityLifecycle Operation: get_company_attributes

```
NameValueList get_company_attributes(
    in wstring company_name)
    raises (UnknownCompany);
```

Description
Returns a list with company attributes associated to the a company name.

Precondition
The client session must have been established with Manager privileges.

Input Parameters
wstring company_name: the name or ID of the company

Output Parameters
none

Return Value
Returns NameValueList: information about a company as a list of name-value pair.

Exceptions
UnknownCompany: raised if trying to get attributes to a non existing company.

Postcondition
none

GLFacilityLifecycle Operation: create_company_chart_of_accounts

```
void create_company_chart_of_accounts(
    in wstring new_company_name,
    in ChartKind chart_of_account_schema,
    in wstring copied_company_name_for_schema)
    raises (UnknownCompany,
    BadChartKind);
```

Description
Creates a new company, and sets up an initial chart of account based on the schema indicated by chart_of_account_schema, or, if copied_company_name_for_schema is non-empty, copies the chart of accounts from that company.

Precondition
The client session must have been established with Manager privileges.

Input Parameters
wstring new_company_name:
ChartKind chart_of_account_schema:
wstring copied_company_name_for_schema:

Output Parameters
none

Return Value
none

Exceptions
UnknownCompany: The company copied_company_name_for_schema does not exist.
BadChartKind: chart_of_account_schema is not a valid ChartKind value.

Postcondition
A ledger for the new company is created and made available to new client sessions.

GLFacilityLifecycle Operation: expunge_company

```
void expunge_company(in wstring company_name)
    raises (UnknownCompany,
    CannotRemove);
```

Description
Deletes the company and its ledger from the Facility.

Precondition
The client session must have been established with Manager privileges.

Input Parameters
wstring company_name: the name of the company

Output Parameters
none

Return Value
none

Exceptions
UnknownCompany: raised if trying to delete a none existing company
CannotRemove: unable to delete the chart of accounts.

Postcondition
none

Other Facility and Company Information

GLFacilityLifecycle Operation: set_GLReport_codes

```
void set_GLReport_codes(in wstringList GLReport_codes)
    raises (BadName, MaxExceeded);
```

Description
Defines the set of reporting codes for accounts in the current ledger.

Precondition
The client session must have been established.

Input Parameters
wstringList GLReport_codes: The valid reporting codes for the ledger.

Output Parameters
none

Return Value
none

Exceptions
BadName: Invalid name. MaxExceeded: Too many reporting codes.

Postcondition
none

GLFacilityLifecycle Operation: set_default_currencies

```
void set_default_currency (
    in wstring currency_mnemonic )
    raises ( BadName);
```

Description
Defines the default currency assigned to new accounts in the current ledger.

Precondition
The client session must have been established with Manager privileges.

Input Parameters
wstring currency_mnemonic: The default currency for new accounts in the ledger.

Output Parameters
none

Return Value
none

Exceptions
BadName: Invalid name.

Postcondition
none

GLFacilityLifecycle Operation: set_known_currencies

```
void set_known_currencies (
    in wstringList currency_mnemonics )
    raises ( BadName, MaxExceeded );
```

Description
Defines the set of valid currencies for transactions posted to the current ledger.

Precondition
The client session must have been established with Manager privileges.

Input Parameters
wstringList currency_mnemonics: The valid currencies for the ledger.

Output Parameters
none

Return Value
none

Exceptions
BadName: Invalid currency name. MaxExceeded: Too many currencies.

Postcondition
none

Appendix A - Requirements Compliance

The scope of the proposed solutions to the RFP is defined by that document in the following statements:

- *this RFP seeks responses that identify the external interfaces, relationships and semantics, that are required for accounting application interoperability with General Ledger (GL) systems.*
- *this RFP does not seek proposals for the internal interfaces of a General Ledger system or other functions that are not required for general interoperability with accounting applications.*
- *this RFP does not seek proposals for other financial and accounting applications, but proposals must define how such other applications could interface and inter-operate with the GL Facility using OMG IDL interfaces.*
- *this RFP is limited exclusively to the General Ledger component of the Common Accounting Facility as found in the OMG Common Facilities Architecture.*

RFP Requirements for the General Ledger Facility

The technical requirements for the General Ledger Facility are specified in the Financial Domain Task Force RFP, document finance/97-11-05, Section 6.0. The following sub-sections follow the outline of that section.

Specific Mandatory Requirements

Requirement
All interfaces of the General Ledger Facility shall be described in OMG IDL and include specification of exception conditions on operations. Proposals shall define the General Ledger Facility in a manner that supports programming language independence and exclude dependence upon specifications that do not provide for implementation independence.

Response
Met in full.
Previous sections of this submission have specified each of the interfaces comprising the proposed General Ledger Facility, including additional exception conditions, using the standard OMG IDL as defined in the CORBA 2.1 specification. Appendix B contains the consolidated IDL specification for the complete set of interfaces. This IDL has been written following the OMG style guidelines and its syntax has been validated using a CORBA 2.1 compliant IDL compiler.

Requirement
Proposals shall provide sufficient level of description to allow for independently developed accounting applications (including legacy) to inter-operate using submitted GL interfaces.

Response
Met in full.
The content of this submission is being used in an on-going European project as the basis for the design of two separate prototype GL implementations, one of which is based on an existing (i.e. legacy) accounting product, which are required to inter-operate at the application level - i.e. semantically.

Requirement
Proposals shall provide GL support for multiple domestic currencies. For example, this requirement derives from the phased transition in the European Union from single indigenous National currencies, to the Euro. For a protracted period, both currencies will be used simultaneously as *domestic* currencies. The combination of US Dollars and UK Sterling is an example of mixed currency support that is not necessarily domestic. General support for multiple international currencies is not mandatory because the vast majority of accounting applications neither require nor implement this capability.

Response
Multiple domestic currencies are handled by the use of multiple sets of Chart of Accounts and multiple sets of transactions. Multiple international currencies are not manipulated by the GL, other than by using the operations inherited from a Currency Facility for Money objects. Any additional handling of such currencies is regarded as the responsibility of the journal application. See Currency Facility dependency below.

Requirement
Proposals shall support GL persistence in a manner that is transparent to accounting applications.

Response
By using the OMG Persistent Object Service (Common Object Services Specification, July 1997) persistence becomes a server-side issue, insulating the client application from these issues.

Requirement
Proposals shall include examples of the behaviour of General Ledger interfaces for clarification.

Response
These are included with the ODP based Enterprise and Information viewpoint documents submitted to OMG along with this submission.

Requirement
Proposals shall provide for several viewpoints of the General Ledger with respect to specific points in time.

Response
The concept of an accounting period is included in this submission and can be used to partitions sets of transactions and/or extracts from the account history. The GLRetrieval interface also supports the ability to extract sets of transaction data from the General Ledger selected by an inclusive date range.

Requirement
Submissions shall support interfaces that enable value added roll-up capabilities, although a full specification of roll-up capabilities is not mandatory and is beyond the scope of the mandatory requirements of this RFP.

Response
The submission allows full retrieval of all transaction data from the General Ledger, which forms the most general solution to supporting an unspecified roll-up capability.

Specific Optional Requirements

Requirement
Proposals may provide for consolidated reporting from multiple General Ledgers. This accounting procedure is often called “roll-up”. Even though roll-up is not required by the majority of accounting users, roll-up is regularly performed in multi-company enterprises (often by manual procedures due to lack of systems integration). If a roll-up capability is submitted, proposals shall address the related systems integration issues.

Response
This submission does not make any specific proposal with regard to roll-up. As previously stated, the submission does provide a very general retrieval capability which could be used by a particular implementation as the basis for adding roll-up functionality.

Requirement
Proposals may provide for localisation of the General Ledger with respect to statutory requirements, natural languages, and local accounting practices.

Response
This submission uses the Money and Date objects from the Currency specification in order to support the use of multiple currencies and date formats. It also requires the use of the IDL *wstring* type for all textual parameters, enabling the use of any supported character set for human-readable data, and avoids specifying any fixed textual data for items such as error or informational messages. The submission is based solely on the accounting principles laid down by the International Accounting Standards Committee, which are accepted and legally required/enforced by over 100 countries world-wide, and does not assume any other accounting practices. This approach minimises restrictions on the ability of specific implementations to support specific accounting regulations and/or practices.

Requirement
Proposals may provide support for multiple textual descriptions of General Ledger entities (such as account names). For example, this capability is desirable in multi-lingual enterprises to support user-selected language preferences. Note that multiple textual descriptions can be manually entered, automated translation is neither necessary nor recommended to support this requirement.

Response
Other than using the *wstring* type for textual items this submission makes no specific provision for multi-lingual support.

Requirement
Proposals may provide support for GL accounts and/or transactions within a single GL implementation to be distributed across multiple servers.

Response
As is normal for any OMG specification, this submission makes no assumptions about the actual GL implementation and places no restrictions on how it may or may not be actually distributed.

Requirement
Proposals may enumerate departments, projects, or other business categories.

Response
In order to maximise its applicability this submission seeks to minimise the number of fixed enumerated categories, preferring to use unbounded lists instead.

Requirement
Proposals may support budgets and budget comparisons.

Response
This submission makes no specific provision for handling budgets as distinct items.

Requirement
Proposals may allow for multiple accounting periods that are "open" simultaneously. For example, a transaction to an accounting period can be made subsequent to the closing date of the accounting period.

Response
This submission makes no prescriptive statements about the rules that may be applied when using accounting periods.

Common Mandatory Requirements

Every RFP issued by the OMG includes a common set of requirements which must be met by every submission (many of these are simply «good practice»).

Requirement
Proposals shall express interfaces in OMG IDL. Proposals should follow accepted OMG IDL and CORBA programming style. The correctness of the IDL shall be verified using at least one IDL compiler (and preferably more then one). In addition to IDL quoted in the text of the submission, all the IDL associated with the proposal shall be supplied to OMG in compiler-readable form.

Response
Met in full.

Requirement
Proposals shall specify operation behaviour, sequencing, and side-effects (if any).

Response
Met in full.

Requirement
Proposals shall be precise and functionally complete. There should be no implied or hidden interfaces, operations, or functions required to enable an implementation of the proposed specification.

Response
Met in full.

Requirement
Proposals shall clearly distinguish mandatory interfaces and other specification elements that all implementations must support from those that may be optionally supported.

Response
Met in full.

Requirement
Proposals shall reuse existing OMG specifications including CORBA, CORBAservices, and CORBAfacilities in preference to defining new interfaces to perform similar functions.

Response
Met in full.

Requirement
Proposals shall justify and fully specify any changes or extensions required to existing OMG specifications. This includes changes and extensions to CORBA inter-ORB protocols necessary to support interoperability. In general, OMG favours upwards compatible proposals that minimise changes and extensions to existing OMG specifications.

Response
This submission contains no such changes or extensions.

Requirement
Proposals shall factor out functions that could be used in different contexts and specify their interfaces separately. Such minimality fosters re-use and avoids functional duplication.

Response
Met in full.

Requirement
Proposals shall use or depend on other interface specifications only where it is actually necessary. While re-use of existing interfaces to avoid duplication will be encouraged, proposals should avoid gratuitous use.

Response
Met in full.

Requirement
Proposals shall specify interfaces that are compatible and can be used with existing OMG specifications. Separate functions doing separate jobs should be capable of being used together where it makes sense for them to do so.

Response
Met in full.

Requirement
Proposals shall preserve maximum implementation flexibility. Implementation descriptions should not be included, however proposals may specify constraints on object behaviour that implementations need to take into account over and above those defined by the interface semantics.

Response
Met in full.

Requirement
Proposals shall allow independent implementations that are substitutable and interoperable. An implementation should be replaceable by an alternative implementation without requiring changes to any client.

Response
Met in full.

Requirement
Proposals shall be compatible with the architecture for system distribution defined in ISO/IEC 10746, Reference Model of Open Distributed Processing (ODP). Where such compatibility is not achieved, the response to the RFP must include reasons why compatibility is not appropriate and an outline of any plans to achieve such compatibility in the future.

Response
Met in full - the RM-ODP has been used extensively during the development of this submission.

Requirement
Proposals shall address relationships to the OMG Security and Transaction Services, whether or not these technologies are utilised

Response
Met in full.

Proof of Concept Statement

The principal contributors to this submission are involved with the EC-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 accounting software vendors, and utilising existing OMG-compliant technology.

The two implementations are based on existing accounting products and demonstrate the applicability of this submission to both legacy and component-based software; they will be available for display at the OMG Technical Meetings following the final submission.

Service Dependencies and Relationships

Security Service
This aspect is currently “work in progress” by the CORBA Security SIG and the FDTF. Security is of paramount importance when dealing with highly sensitive financial information. One approach may be to specify some application-level security capabilities, dealing with authorisation and access control, which could be implemented either by using the existing Security Service (at level one or above) or by the application itself if no Security Service implementation is available.

Object Transaction Service (OTS)
While this submission makes no explicit use of the Transaction Service, it is likely that implementations targeted at large enterprises will take advantage of the facilities of this Service for scalability.

Unified Modelling Language (UML)
The underlying models derived by the COMPASS partners as part of their work on the design of the interface structure are based on the RM-ODP approach and documented using UML. See Volume II of this submission.

Calendar Facility
This submission makes no explicit use of this Facility.

Currency Facility
This submission uses a FdCurrency Facility which provides several opaque types to this specification. As some specifications and features are not yet finalised by OMG nor available in actual implementations, the submitters have compiled the GL IDL by:

- *changing value types to interface types*
- *use placeholder versions of any non-available service types in the appropriate included IDL file*

Workflow Facility
This submission makes no explicit use of the Workflow Facility.

Time and Internationalisation Facility
This submission makes no explicit use of this Facility.

Event Service
This submission makes no explicit use of the Event Service.

Pass-by-Value
Although this submission does not explicitly require support for Pass-by-Value, it is used by the Currency Facility.

Notification
This submission makes no explicit use of the Notification Service.

Party Management Facility
This submission makes no explicit use of this Facility.

Relationship Service
Although this submission does not use this service itself, it is used in the Currency Facility (see above).

Query Service
Although this submission does not use this service itself, it is used in the Currency Facility (see above).

Persistence Service
Although this submission makes no explicit use of this service, its use is assumed in order to provide server-side persistence.

Messaging Service
This submission may be updated to take advantage of the improved semantics provided by this service for *one-way* operations; the optional routing part of this service will not be included.

For convenience, the status of the above, and other relevant technologies, within the OMG process is shown below. Note: last checked against OMG web site on 29th March 1998.

Current stage in the OMG process	Technology
Formally approved	Unified Modelling Language (UML)
	Event Service
	Time and Internationalisation
Formal adoption vote completed	COM/CORBA Part B
	Object Pass-by-Value
	CORBA Core RTF
	ORB Interoperability RTF
	C++ Mapping RTF
	Security 1.2 RTF
Adoption vote in progress	Currency Facility
Revised submission(s) received	Notification
	Workflow Facility
	Business Objects
Initial submission(s) received	CORBA Component Model
	Party Management Facility
Awaiting initial submission(s)	Calendar Facility
Awaiting RFI responses	Common Business Objects

Relationship to CORBA

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

Relationship to the OMG Object Model

The General Ledger Facility conforms to the OMG Object Model.

Appendix B - General Ledger Facility IDL

```
#include <FdCurrency.idl>

module FdGeneralLedger {

    // FORWARD DECLARATIONS

    interface GLProfile;                // establish client session
    interface GLBookKeeping;            // data entry
    interface GLRetrieval;              // data acquisition
    interface GLIntegrity;              // data integrity checks
    interface GLAccountLifecycle;       // GL Account lifecycle management
    interface GLFacilityLifecycle;      // GL Facility lifecycle management

    // DATA TYPE DECLARATIONS

    typedef sequence<boolean> booleanList;
    typedef sequence<wstring> wstringList;

    struct NameValue {
        wstring name;
        wstring value; };                   // TBD
    typedef sequence<NameValue> NameValueList;

    typedef FdCurrency::Date Date;
    typedef FdCurrency::Money Money;
    typedef wstring Currency;               // ISO CURRENCY MNEMONIC

    enum ChartKind {DEFAULT_NOMINAL, EXISTING_CHART, EMPTY_LEDGER };

    struct AccountInfo {
        wstring acc_ref;
        wstring description; };
    typedef sequence<AccountInfo> AccountInfoList;

    enum AccountKind { CASH, BANK, CONTROL, REGULAR };

    struct Account {
        wstring         GLAcc_ref;          // GL Account reference
        wstring         GLAcc_name;         // name
        wstring         GLreporting_code;
        Currency        default_currency;
        boolean         is_control;
        Money           tp_bal;
        Money           ytd_bal;
        wstring         con_acc_kind;
        wstring         con_acc_desc;
        wstringList     optional_fields; };
    typedef sequence<Account> AccountList;

    enum PeriodKind { NO_DATE, WEEK, MONTH, QUARTER, YEAR };

    struct AccountingPeriod {
        wstring period_name;
        PeriodKind period_kind;
        Date start_date;
        Date end_date; };
```

```
    struct HistorySpec{
        wstring lower_acc_ref, upper_acc_ref;
        wstring start_period, end_period;
        Date start_date, end_date;
        wstring lower_trans_no, upper_trans_no; };
    typedef sequence<HistorySpec> HistorySpecList;

    struct TransactionInfo {
        wstring         trans_no;
        wstring         trans_kind;
        wstring         period_id;  // TBD
        Date            trans_date; };
    typedef sequence <TransactionInfo> TransactionInfoList;

    struct Entry {
        unsigned long   trans_no;
        Date            entered_date;
        wstring         account_no;
        wstringList     dimension_accounts;
        Money           amount;
        double          quantity;
        wstring         description;
        wstring         rule_ref;
        wstring         invoice_no;
        wstring         document_ref;
        wstring         user_name;
        NameValueList         optional_fields; };
    typedef sequence<Entry> EntryList;

    struct Transaction {
        unsigned long   trans_no;
        wstring         trans_kind;
        wstring         period_id;
        Date            trans_date;
        wstring         document_ref;
        EntryList       entries;
        NameValueList         optional_fields; };
    typedef sequence<Transaction> TransactionList;

    // EXCEPTION DECLARATIONS

    exception BadDate { wstring error;
        Date bad_value; };
    exception BadChartKind { wstring error;
        ChartKind bad_value; };
    exception BadSelection { wstring error;
        unsigned long selection_code;
        booleanList bad_members; };
    exception BadTransaction { wstring error;
         wstring trans_no; booleanList bad_fields; };
    exception BadAccountKind {wstring error;
        AccountKind bad_value; };
    exception BadHistorySpec { wstring error;
        booleanList bad_members; };
   exception BadPeriod {
       wstring error;
       wstring period_id; };              // TBD.
   exception BadName { wstring error, bad_value; };
   exception BadAccountRef { wstring error;
       wstring bad_value; };
   exception BadTransNo { wstring error;
```

```
        wstring bad_value; };
    exception NoChartOfAccounts { wstring error; };
    exception CannotRemove { wstring error; };
    exception ProfileError { wstring error; };          // TBD.
    exception UnknownCompany { wstring error, bad_value; };
    exception MaxExceeded { wstring error;
        unsigned long max_amount; };
    exception BadIntegrity { wstring error;
        any bad_value; };
    exception BadAccountName { wstring error;
        wstring bad_value; };
    exception BadReportingCode { wstring error;
        wstring bad_value; };

     // INTERFACE DECLARATIONS

     interface GLProfile {

         // PROFILE OPERATIONS
         wstring get_default_company_name() raises ( NoChartOfAccounts );
         wstringList get_GL_company_names() raises ( NoChartOfAccounts );
         Date get_current_system_date();
         wstring general_ledger_open (
              in wstring company_name, in Date system_date )
              raises ( UnknownCompany, ProfileError, BadDate );
         void quit() raises ( ProfileError );

         // FRAMEWORK OPERATIONS
         GLBookKeeping book_keeping() raises ( ProfileError );
         GLRetrieval retrieval() raises ( ProfileError );
         GLIntegrity integrity() raises ( ProfileError );
         GLAccountLifecycle account_lifecycle() raises ( ProfileError );
         GLFacilityLifecycle facility_lifecycle() raises ( ProfileError );

         // PROFILE INFORMATION
         wstring get_client_company_name() raises ( ProfileError,
NoChartOfAccounts );
         AccountingPeriod get_current_period() raises ( ProfileError );
         wstringList get_tax_codes() raises ( ProfileError );      //TBD.
         wstringList get_GLReport_codes() raises ( ProfileError );
         wstring get_default_currency() raises ( ProfileError );
         wstringList get_known_currencies() raises ( ProfileError );
         wstringList get_dimension_names() raises ( ProfileError );
     };

     interface GLRetrieval {

         // ACCOUNT RETRIEVAL
         unsigned long number_of_accounts();
         AccountInfoList get_account_info ( in AccountKind type_of_account )
             raises ( BadAccountKind );
         AccountInfoList get_all_account_info();
         Account get_account ( in wstring GLAcc_ref )
             raises ( BadAccountRef );
         AccountList get_multiple_accounts ( in wstringList account_refs )
             raises ( BadAccountRef );
         AccountList get_accounts_from_GLreporting_code(
             in wstring GLreporting_code)
             raises (BadReportingCode);
         AccountInfoList get_control_acc_info();
```

```
        // TRANSACTION RETRIEVAL
        unsigned long number_of_current_transactions();
        TransactionInfo get_transaction_info(in wstring trans_no) raises
(BadTransNo);
        TransactionInfoList get_multiple_transaction_info (
            in HistorySpec history_range )
            raises ( BadHistorySpec, MaxExceeded );
        HistorySpec get_current_history_range (
            out unsigned long number_of_transactions );
        unsigned long number_of_history_transactions (
            in HistorySpec history_range )
            raises ( BadHistorySpec );
        Transaction get_transaction(in wstring trans_no)
             raises (BadTransNo);
        TransactionList get_multiple_transactions ( in HistorySpec history_range
)
            raises ( BadHistorySpec, MaxExceeded );
    };

    interface GLBookKeeping {

        void post ( in Transaction single_transaction ) raises ( BadTransaction
);
        void post_batch ( in TransactionList transactions ) raises (
BadTransaction );

    };

    interface GLAccountLifecycle {

        // ACCOUNT LIFECYCLE
        void createAccount(in wstring GLAcc_ref,
            in wstring GLAcc_name,
            in boolean is_nominal_account,
            in boolean is_control_account,      // TBD.
            in wstring reporting_code)
            raises (BadAccountRef, BadAccountName, BadReportingCode);
        void removeAccount(in wstring GLAcc_ref)
            raises (BadAccountRef, CannotRemove);
        void modify_account(in wstring GLAcc_ref,
            in wstring new_GLAcc_name,
            in wstring new_reporting_code)
            raises (BadAccountRef, BadAccountName, BadReportingCode);

        // PERIOD/YEAR CLOSING
        void close_accounting_period(
            in wstring period_id)
            raises (BadPeriod);
        void close_accounting_year(
            in wstring last_period_in_year)
            raises(BadPeriod);

    };

    interface GLIntegrity {

        wstringList get_dynamic_selection();
        boolean check_integrity ( in wstring integrity_check_selection )
            raises (BadSelection, BadIntegrity );

    };
```

```
    interface GLFacilityLifecycle {

        // COMPANY/CHART OF ACCOUNTS LIFECYCLE
        NameValueList get_company_attributes ( in wstring company_name )
            raises ( UnknownCompany );
        void create_company_chart_of_accounts ( in wstring new_company_name,
            in ChartKind chart_of_account_schema,
            in wstring copied_company_name_for_schema )
            raises ( UnknownCompany, BadChartKind );
        void expunge_company ( in wstring company_name )
            raises ( UnknownCompany, CannotRemove );
        void set_GLReport_codes ( in wstringList GLReport_codes )
            raises ( BadName, MaxExceeded );
        void set_default_currency ( in wstring currency_mnemonic )
            raises ( BadName );
        void set_known_currencies ( in wstringList currency_mnemonics )
           raises ( BadName, MaxExceeded );

    };

};                                  // end of FdGeneralLedger
```

Appendix C - References

[1] Executive Encyclopaedia: 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 Standardisation: Theory, Process and Organisations: 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 modelling 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 Standard," 1997.

[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 Behavioural Semantic," Institut fur Informatik der Technischen Universitat Munchen 1997.

[27] MAGMA, "Magma Software engineering handbook," SINTEF, draft 1997.

[28] C. R. Malburg, Accounting for a new business: Bob Adams Inc, 1994.

[29] L. v. Mises, Human: Action: A Treatise on Economics: Regnery, 1963.

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

[31] T. J. Mowbray and R. C. Malveau, CORBA Design Patterns: John Wiley & Sons, Inc., 1997.
[32] T. J. Mowbray and W. A. Ruh, Inside CORBA: John Wiley & Sons, 1997.
[33] T. J. Mowbray and R. Zahavi, The Essential CORBA: Systems Integration Using Distributed Objects: John Wiley & Sons, Inc., 1995.
[34] OMG, "Common Facilities RFP3," OMG Document Number 95.11.3, November 1995.
[35] OMG, "CorbaFacilities: Common Facilities Architecture," Object Management Group Revision 4.0, November 1995.
[36] OMG, "OMG Object Management Architecture Guide (OMA Guide), Revision 3.0," , 1995.
[37] OMG, "CORBAservices: Common Object Services Specification," , 1997.
[38] OMG, "The Common Object Request Broker: Architecture and Specification, Revision 2.2," Object Management Group Feb. 1998.
[39] OMG/UML, "UML Notation," . http://www.rational.com/uml/html/notation, 1997.
[40] OMG/UML, "UML Semantics," . http://www.rational.com/uml/html/semantics, 1997.
[41] R. Orfali and D. Harkey, Client/Server Programming with Java and CORBA: John Wiley & Sons, Inc., 1997.
[42] T. Reenskaug, P. Wold, and O. A. Lehne, Working with Objects - The OOram Software Engineering Method: Manning Publications, ISBN 1-884777-10-4, 1996.
[43] 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.
[44] J. D. Shank and V. Govindarajan, "Strategic Cost Management and the Value Chain," Journal of Cost Management, vol. 5, pp. pp 5-21, 1992.
[45] M. Shaw and D. Garlan, Software Architecture - Perspectives On An Emerging Discipline: Prentice-Hall, 1996.
[46] J. Siegel, CORBA Fundamentals and Programming: John Wiley & Sons, 1997.
[47] C. Szyperski, Component Software, Beyond Object-Oriented Programming: Addison-Wesley, 1998.
[48] P. B. B. Turney, Common Cents: The ABC Performance Breakthrough: Hillsboro, 1991.