IONA®

Orbix®

First Northern Bank Demo Mainframe Guide

Version 6.0, November 2003

Making Software Work Together™

Updated: 15-Dec-2003

M 3 1 7 0

Contents

Preface

Orbix provides a demonstration called First Northern Bank (FNB) that integrates CORBA, J2EE, and Web services components. This guide is intended for use when running the FNB demonstration with the FNB COBOL back-end server supplied with Orbix Mainframe. It provides an introductory overview of the entire FNB demonstration in terms of the technologies it supports, but focuses specifically on the development and running of the FNB COBOL back-end server.

This document is intended as an addendum or complement to the core FNB documentation set that is supplied with Orbix. For full details of the development and management of the front-end and middle-tier components of the FNB demonstration, see the core FNB documentation set at `http://www.iona.com/support/docs/orbix/6.1/tutorials.xml`.

If you need help with this or any other IONA products, contact IONA at `support@iona.com`. Comments on IONA documentation can be sent to `docs-support@iona.com`.

Audience

Chapter 1 is intended for anyone who wants to familiarize with the overall architecture of the FNB demonstration.

Chapters 2 and 3 are intended for COBOL application programmers who want to develop and run CORBA applications on OS/390. The prerequisites are a good knowledge of COBOL and familiarity with basic CORBA concepts. See the *Mainframe Concepts Guide* for more details of basic CORBA concepts.

Organization of this guide

This guide is divided as follows:

Chapter 1, Introduction

This chapter introduces the overall FNB demonstration architecture, and its CORBA, J2EE, and Web services components.

Chapter 2, "Developing the FNB COBOL Back-End Server"

This chapter discusses the design and implementation of the COBOL back-end server component of the FNB demonstration. The server is implemented in COBOL and runs in batch on OS/390.

Chapter 3, Running the FNB COBOL Back-End Server

This chapter describes how to start the COBOL back-end server component of the FNB demonstration.

Related documentation

The following Orbix documentation provides details of the development and management of the front-end and middle-tier components of the FNB demonstration:

- *First Northern Bank Tutorial*
- *First Northern Bank Developer's Introduction*

These documents can be found at `http://www.iona.com/support/docs/orbix/6.1/tutorials.xml`.

The *COBOL Programmer's Guide and Reference* supplied with Orbix Mainframe complements this guide by providing more details of CORBA application development in COBOL on OS/390.

The latest updates to all Orbix Mainframe documentation can be found at `http://www.iona.com/support/docs/orbix/mainframe/6.0/index.xml.`

Additional resources

The IONA knowledge base contains helpful articles, written by IONA experts, about the Orbix and other products. You can access the knowledge base at the following location:

`http://www.iona.com/support/kb/`

The IONA update center contains the latest releases and patches for IONA products:

`http://www.iona.com/support/update/`

Typographical conventions

This guide uses the following typographical conventions:

`Constant width`	Constant width (courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `CORBA::Object` class. Constant width paragraphs represent code examples or information a system displays on the screen. For example: `#include <stdio.h>`
Italic	Italic words in normal text represent *emphasis* and *new terms*. Italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example: `% cd /users/`*your_name* **Note:** Some command examples may use angle brackets to represent variable values you must supply. This is an older convention that is replaced with *italic* words or characters.

Keying conventions

This guide may use the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, a prompt is not used.
%	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
#	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
>	The notation > represents the DOS, Windows NT, Windows 95, or Windows 98 command prompt.
`...` . . .	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.

[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices enclosed in { } (braces) in format and syntax descriptions.

CHAPTER 1

Introduction

Orbix provides a demonstration called First Northern Bank (FNB) that integrates CORBA, J2EE, and Web services components. This guide is intended for use when running the FNB demonstration with the FNB COBOL back-end server that is supplied with Orbix Mainframe. This chapter introduces the overall FNB demonstration architecture, and its CORBA, J2EE, and Web services components.

In this chapter

This chapter discusses the following topics:

First Northern Bank Architecture

Overview

This section describes the high-level architecture of the new FNB system, and gives a brief overview of its components. It includes the following topics:

- "FNB architecture".
- "CORBA banking".
- "J2EE internet banking".
- "Web services credit card validation".

FNB architecture

Figure 1 shows the overall FNB demonstration system architecture.

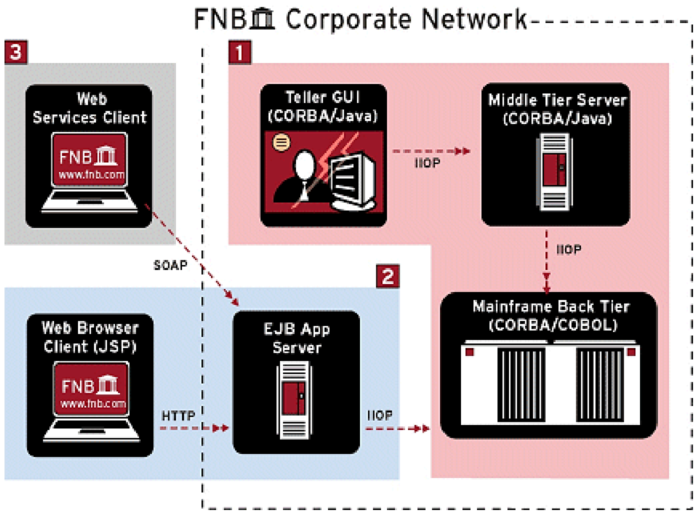

Figure 1: *FNB System Architecture*

The main components in Figure 1 are as follows:

1. CORBA core banking.
2. J2EE Internet banking.

3. Web services credit card validation.

CORBA banking

The CORBA banking application provides the core banking services that the bank offers to its customers. For example, opening an account, making a deposit, or making a withdrawal.

The CORBA banking application is implemented as a three-tier system, which consists of the following components:

- Bank teller client GUI (Graphical User Interface) on Windows.
- Middle-tier Java server on Windows or UNIX.
- Back-tier COBOL server on OS/390.

All network communication is sent using the Internet Inter-ORB Protocol (IIOP).

J2EE internet banking

The J2EE application provides customers with Internet banking services. It provides Web browser access to customer accounts (for example, viewing an account balance, or paying a bill online).

The J2EE Internet banking application is implemented using Enterprise Java Beans (EJBs) and Java Server Pages (JSPs), which run in the Orbix Application Server. This in turn connects to the back-tier CORBA server on OS/390.

Network communications between the application server and the browser client are sent using the Hypertext Transfer Protocol (HTTP). Those between the application server and the back-tier OS/390 server are sent using IIOP.

Web services credit card validation

The Web services application provides online credit card validation and payment services for customers. Figure 1 shows a Web services client application that invokes on the Web service running in the application server. This client could be implemented in several programming languages (for example, Java, C#, or Visual Basic).

Like the CORBA and J2EE systems, the Web services application is also a three-tier system. Network communications between the Web service and client are sent using the Simple Object Access Protocol (SOAP) over HTTP.

CORBA Banking Application

Overview

This section describes the CORBA core banking application in more detail. It includes the following topics:

- "CORBA bank architecture".
- "Bank teller GUI client".
- "Middle-tier CORBA server".
- "Back-tier CORBA server".

CORBA bank architecture

Figure 2 shows the architecture of the three-tier CORBA banking application.

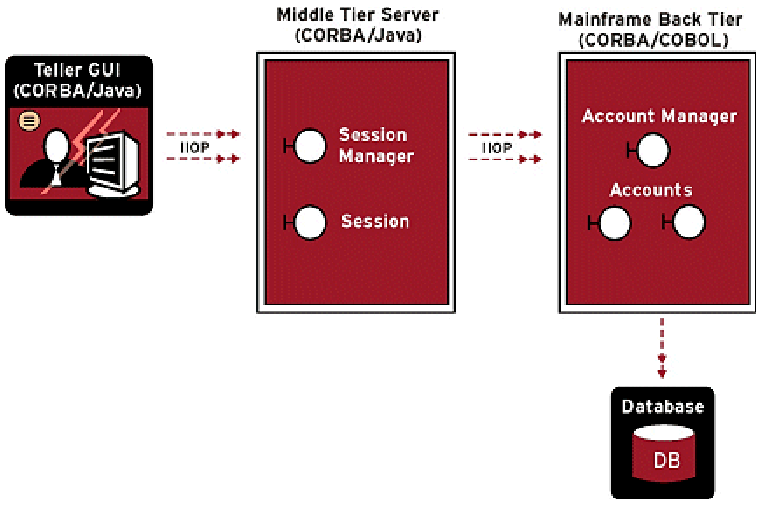

Figure 2: *FNB Bank Application*

The main components in Figure 2 are as follows:

- Front-tier client used by bank teller (Java GUI).
- Middle-tier business architecture (CORBA Java server).
- Back-tier mainframe system (CORBA COBOL server) using VSAM files.

Bank teller GUI client

The bank teller GUI enables tellers to open and close accounts, and to make withdrawal and lodgements to accounts. The bank teller GUI is implemented as a Java Swing client application.

Middle-tier CORBA server

The middle-tier CORBA Java server manages business sessions between the client and the back-tier server.

The middle-tier server implements a `BussinesSessionManager` factory object, which creates session objects to manage interaction with the client (for example, `TellerSession` and `BusinessSession` objects).

The middle-tier CORBA server is also known as the FNB Business Architecture (FNBBA).

Back-tier CORBA server

The back-tier CORBA server on the mainframe is responsible for managing customer accounts. This is implemented as the COBOL `FNB` server. The back-end server is automatically deployed on OS/390 when you install Orbix Mainframe.

The `FNB` server implements an `AccountMgr` factory object, which creates `Account` objects (for example, `CreditCardAccount` and `CurrentAccount` objects). These objects represent all customer account information (for example, customer name, address, and account number).

The `Account` objects are stored in VSAM data sets on OS/390. Four VSAM data sets are used, to store the following data:

- Account data—this includes an alternative index, to allow for referencing data by account number or account type.
- Transaction history.
- Last used account number.
- Last used transaction history key (for each account).

Note: For details on development and building of the COBOL back-end server, see "Developing the FNB COBOL Back-End Server" on page 11.

J2EE Internet Banking Application

Overview

This section describes the J2EE Internet banking application and its components in more detail. It includes the following topics:

- "Internet banking architecture".
- "Web browser client".
- "J2EE application server".
- "Cloudscape database".
- "Back-tier CORBA server".

Internet banking architecture

Figure 3 shows the architecture of the three-tier J2EE Internet banking application.

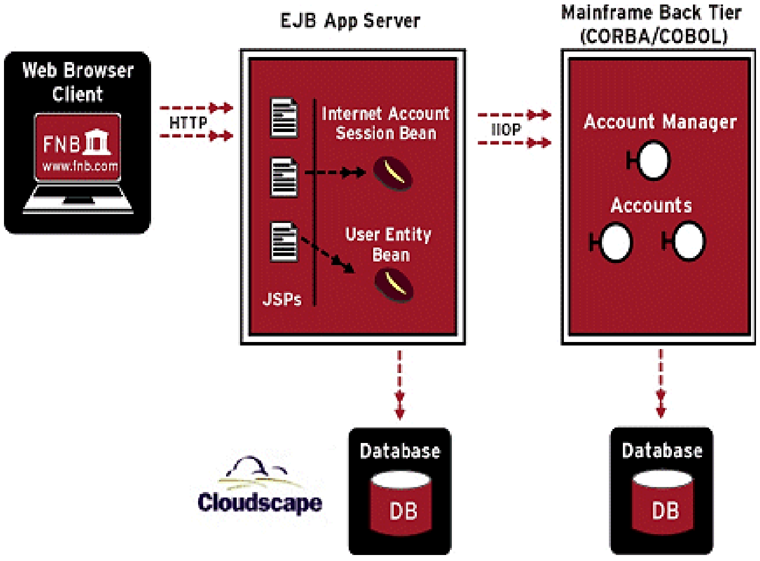

Figure 3: *FNB Internet Banking Application*

The main components in Figure 3 are as follows:

- Web browser client on Windows.
- Middle-tier J2EE application server on Windows or UNIX.
- Cloudscape database.
- Back-tier CORBA COBOL server on OS/390 using VSAM files.

Web browser client

A standard Web browser provides Internet banking services to customers. Users must first register, and create a user ID and password, before logging on. Internet banking services include viewing an account balance and paying a bill online.

Network communications between the Web browser and the application server are sent using HTTP.

J2EE application server

An Orbix J2EE application server provides the middle-tier J2EE infrastructure. It runs the Java Server Pages (JSPs) that serve up the Internet banking Web pages in the browser. The application server also runs the Enterprise Java Beans (EJBs) that communicate with the Cloudscape database and the back-tier CORBA server on OS/390.

For example, the `User` entity bean handles the customer information stored in the database; while the Internet account session bean (`inetAccount`) handles browser sessions with the back-tier server.

Cloudscape database

A Cloudscape database stores customer information that is used to access customer accounts online (for example, the user ID and password associated with each customer account).

Back-tier CORBA server

Communications between the application server and the back-tier server are sent using IIOP. See "CORBA Banking Application" on page 4 for more information about the back-tier CORBA server.

Note: For details on development and building of the COBOL back-end server, see "Developing the FNB COBOL Back-End Server" on page 11.

Web Services Credit Card Validation Application

Overview

This section describes the Web services credit card application and its components in more detail. It includes the following topics:

- "Credit card validation architecture".
- "J2EE application server".
- "J2EE application server".
- "Orbix XMLBus".
- "Back-tier CORBA server".

Credit card validation architecture

Figure 4 shows the architecture of the three-tier credit card validation application.

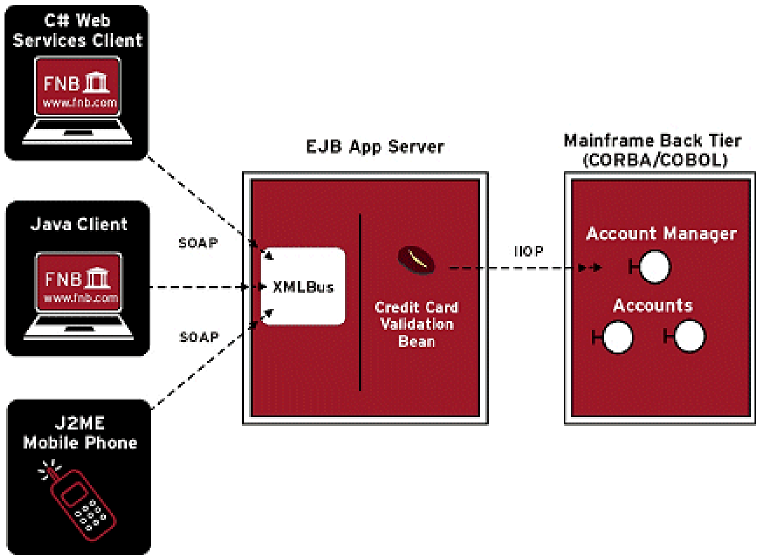

Figure 4: *FNB Credit Card Validation Application*

The main components in Figure 4 are as follows:

- Web services clients on Windows.
- Middle-tier J2EE application server on Windows or UNIX.
- Orbix XMLBus.
- Back-tier CORBA COBOL server.

Web services clients

The Web services client applications provide online facilities for credit card validation and confirmation of purchase.

Figure 4 shows a variety of client applications. Because this is a Web service, the client could be written in several programming languages (for example, C#, Java, or Visual Basic). This tutorial demonstrates how to use a Web services test client that is provided by Orbix XMLBus, IONA's Web services environment.

J2EE application server

The middle-tier Orbix application server runs the Orbix XMLBus Web services environment, and the `ValidateCreditCard` session bean, shown in Figure 4. The application server communicates with the Web services client using SOAP and HTTP.

See "J2EE Internet Banking Application" on page 6 for more information about the application server.

Orbix XMLBus

Orbix XMLBus is IONA's Web services environment. In the First Northern Bank demonstration, the XMLBus version supplied with the Orbix Application Server Platform runs in an Orbix Application Server. The application server forwards the HTTP request to the XMLBus Container, which decodes and handles the incoming SOAP message.

You can also run Orbix XMLBus in other application server environments (for example, IBM WebSphere, BEA WebLogic, and Apache Tomcat).

Back-tier CORBA server

Communication between the application server and the back-tier server is sent using IIOP. See "CORBA Banking Application" on page 4 for more information about the back-tier CORBA server.

Note: For details on development and building of the COBOL back-end server, see "Developing the FNB COBOL Back-End Server" on page 11.

CHAPTER 2

Developing the FNB COBOL Back-End Server

This chapter discusses the design and implementation of the COBOL back-end server component of the First Northern Bank (FNB) demonstration. The server is implemented in COBOL and runs in batch on OS/390.

In this chapter

This chapter discusses the following topics:

Note: For more details about CORBA application development in COBOL on OS/390 see the *COBOL Programmer's Guide and Reference*. For more details of the development of the front-end and middle-tier components of the FNB demonstration see the *First Northern Bank Developer's Introduction* supplied on the Orbix Documentation CD, or online at: http://www.iona.com/support/docs/orbix/6.1/tutorials.xml.

Introduction

Overview

This section introduces the COBOL back-end server component of the FNB demonstration in terms of its purpose and design. It also outlines where you can find the various source code and JCL elements for it.

In this section

This section discusses the following topics:

Purpose and Design

Overview

This subsection provides an overview of the purpose and design of the COBOL back-end server component of the FNB demonstration. It discusses the following topics:

- "Purpose" on page 14.
- "CORBA object types" on page 15.
- "AccountMgr object" on page 15.
- "Account objects" on page 15.
- "CurrentAccount objects" on page 16.
- "CreditCardAccount objects" on page 16.

Purpose

The purpose of the back-end server is to provide the basic business objects for the bank application—in this demonstration, `Account` objects. It accepts and processes requests from the middle-tier FNB Business Architecture across the network.

The back-end server has the following general characteristics:

- Provides close integration with persistent storage—the CORBA back-end server consists of a wrapper around a database that stores the business data.
- Provides an implementation of `Account` CORBA objects—the account data thus becomes accessible to other distributed applications.
- Ignores presentation requirements—the back-end server is not concerned with the way in which clients access and use the `Account` objects. This is left to other parts of the distributed application.

CORBA object types

Figure 5 shows the inheritance hierarchy for the object types implemented in the COBOL back-end server. There is a corresponding interface of the same name defined for each of the object types shown (see “Developing the Application Interfaces” on page 21 for more details).

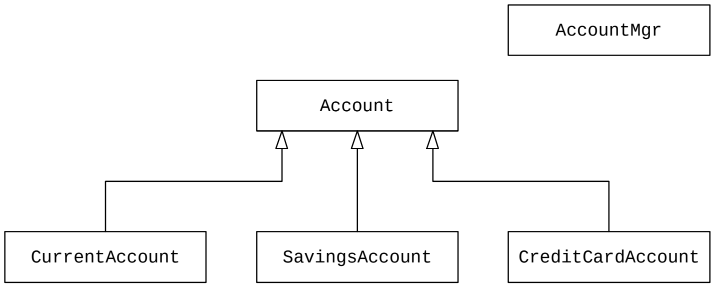

Figure 5: *Inheritance Hierarchy for Account Object Types*

Note: This version of the FNB COBOL back-end server does not implement the `SavingsAccount` objects.

AccountMgr object

A single `AccountMgr` factory object is created, based on the `AccountMgr` interface. A factory object is an object that creates instances of other object types. The `AccountMgr` factory object is used to manage and provide access to the `Account` objects. The `AccountMgr` factory object is needed to:

- Create new `Account` objects.
- Find existing `Account` objects—two alternative search methods are supported:
    - Lookup by account number.
    - Listing all accounts of a particular type.

Account objects

As shown in Figure 5 on page 15, the `Account` interface is (in CORBA terms) an *abstract base interface* from which other, *concrete* interfaces derive. An abstract base interface is not used directly to implement CORBA objects. Instead, the interfaces that derive from the base interface inherit all the elements of it. Therefore, an object that implements a derived interface can accept invocations on any of the elements of the derived interface and the base interface.

A number of attributes are defined on the base `Account` interface:

- Account number.
- Owner details (name and address).
- A list of recent transactions.

Methods are also defined on the base `Account` interface, as follows:

- Deposit and withdraw cash.
- Transfer money in or out of the account.

CurrentAccount objects

Any `CurrentAccount` object is based on the `CurrentAccount` interface. The following attribute is defined on the `CurrentAccount` interface:

- Current overdraft limit.

The following method is also defined:

- Request approval for a new overdraft limit.

Because the `CurrentAccount` interface derives from the `Account` interface, any `CurrentAccount` object can accept invocations on all the attributes and methods of both the `CurrentAccount` and `Account` interface.

CreditCardAccount objects

Any `CreditCardAccount` object is based on the `CreditCardAccount` interface. The following attributes are defined on the `CreditCardAccount` interface:

- Credit limit.
- Interest rate on overdue payments.

The following methods are also defined:

- Authorize an amount of money to be spent.
- Make a purchase, based on an authorization code.
- Calculate the interest due on late payments.

Because the `CreditCardAccount` interface derives from the `Account` interface, any `CreditCardAccount` object can accept invocations on all the attributes and methods of both the `CreditCardAccount` and `Account` interface.

COBOL and interface inheritance

COBOL for OS/390 does not support the concept of IDL interface inheritance. To cater for this, and to avoid having to duplicate code in the implementation of all methods that are inherited from the base `Account` interface, the FNB COBOL server implementation implements each base interface method only once, and has the derived interface methods calling the implemented base methods by means of `PERFORM` statements.

This should not be seen as a standard or even recommended way of overcoming interface inheritance restrictions within COBOL, but it is one possible tradeoff between theory and common sense design in a language that does not support interface inheritance on OS/390.

This approach to implementing the FNB server, however, works only under the premise that the base `Account` methods should not be called directly by the client. For this reason, the implementation code for any base method does not include direct calls to `COAGET` or `COAPUT`. See "Writing the Server Implementation" on page 38 for more details. See the preface of the *COBOL Programmer's Guide and Reference* for details of supported compilers.

Location of Supplied Elements

Overview

All the source code and JCL components needed to create and run the COBOL back-end server for the FNB demonstration have been provided with your Orbix Mainframe installation. This subsection provides an overview of these components. It discusses the following topics:

- "Location of supplied code and JCL" on page 18.
- "Location of supplied copybooks" on page 19.

Location of supplied code and JCL

Table 1 provides a summary of the supplied code elements and JCL components that are relevant to the `FNB` COBOL demonstration (where *`orbixhlq`* represents your installation's high-level qualifier). Apart from site-specific changes to some JCL, these do not require editing.

Table 1: *Supplied Code and JCL (Sheet 1 of 2)*

Location	Description
`orbixhlq``.DEMOS.IDL(FNB)`	This is the supplied IDL for the FNB server.
`orbixhlq``.DEMOS.IDL(DATADEFS)`	This is supplied IDL that defines some basic data types used by the FNB server.
`orbixhlq``.DEMOS.COBOL.SRC(FNBSV)`	This is the source code for the FNB server mainline module.
`orbixhlq``.DEMOS.COBOL.SRC(FNBS)`	This is the source code for the FNB server implementation module.
`orbixhlq``.JCL(LOCATOR)`	This JCL runs the Orbix locator daemon.
`orbixhlq``.JCL(NODEDAEM)`	This JCL runs the Orbix node daemon.
`orbixhlq``.JCL(NAMING)`	This JCL runs the Orbix naming service.
`orbixhlq``.DEMOS.COBOL.BLD.JCL(FNBIDL)`	This JCL runs the Orbix IDL compiler, to generate COBOL copybooks for the FNB server. The `-S` and `-Z` compiler arguments, which generate server mainline and server implementation code respectively, are disabled by default in this JCL.

Table 1: *Supplied Code and JCL (Sheet 2 of 2)*

Location	Description
`*orbixhlq*.DEMOS.COBOL.BLD.JCL(NAMESIDL)`	This JCL runs the Orbix IDL compiler, to generate COBOL copybooks for the IDL operations defined in the `*orbixhlq*.INCLUDE.OMG.IDL(COSNAMI)` IDL member for the Naming Service.
`*orbixhlq*.DEMOS.COBOL.BLD.JCL(FNBSB)`	This JCL compiles and links the batch server mainline and batch server implementation modules.
`*orbixhlq*.DEMOS.COBOL.BLD.JCL(FNBVSAMP)`	This JCL generates offline prints of the VSAM data sets used by the FNB demonstration. This is not necessary for running the FNB demonstration. You can submit this JCL if you want to print the contents of the VSAM files. **Note:** This job yields a return code of `12` if the FNB server is active. This is expected behavior, because the server should not be active while running this job.
`*orbixhlq*.DEMOS.COBOL.RUN.JCL(FNBSV)`	This JCL runs the server.

Location of supplied copybooks

Table 2 provides a summary in alphabetic order of the various copybooks supplied with your Orbix Mainframe installation that are relevant to this batch server demonstration. Again, *orbixhlq* represents your installation's high-level qualifier.

Table 2: *Supplied Copybooks (Sheet 1 of 2)*

Location	Description
`*orbixhlq*.INCLUDE.COPYLIB(CHKERRS)`	This contains a COBOL paragraph that can be called to check if a system exception has occurred, and to report that system exception.
`*orbixhlq*.INCLUDE.COPYLIB(CHKFILE)`	This is used for file handling error checking.
`*orbixhlq*.INCLUDE.COPYLIB(CORBA)`	This contains various Orbix COBOL definitions, such as `REQUEST-INFO` used by the `COAREQ` function, and `ORBIX-STATUS-INFORMATION` which is used to register and report system exceptions raised by the COBOL runtime.

Table 2: *Supplied Copybooks (Sheet 2 of 2)*

Location	Description
`orbixhlq.INCLUDE.COPYLIB(CORBATYP)`	This contains the COBOL typecode representation for IDL basic types.
`orbixhlq.INCLUDE.COPYLIB(FNBACCNO)`	This is specific to the FNB demonstration. It defines the layout of the account number records used in this demonstration.
`orbixhlq.INCLUDE.COPYLIB(FNBRECS)`	This is specific to the FNB demonstration. It defines the layout of the account records, transaction account history records, and transaction number records used in this demonstration.
`orbixhlq.INCLUDE.COPYLIB(IORFD)`	This contains the COBOL `FD` statement entry for file processing, for use with the `COPY…REPLACING` statement.
`orbixhlq.INCLUDE.COPYLIB(IORSLCT)`	This contains the COBOL `SELECT` statement entry for file processing, for use with the `COPY…REPLACING` statement.
`orbixhlq.INCLUDE.COPYLIB(PROCPARM)`	This contains the appropriate definitions for a COBOL program to accept parameters from the JCL for use with the `ORBARGS` API (that is, the `argument-string` parameter).
`orbixhlq.DEMOS.COBOL.COPYLIB`	This PDS is used to store all batch copybooks generated when you run the JCL to run the Orbix IDL compiler for the supplied demonstrations. It also contains copybooks with Working Storage data definitions and Procedure Division paragraphs for use with the bank, naming, and nested sequences demonstrations.

Checking JCL components

When creating the `FNB` COBOL back-end application, check that each step involved within the separate JCL components completes with a condition code of zero. If the condition codes are not zero, establish the point and cause of failure. The most likely cause is the site-specific JCL changes required for the compilers. Ensure that each high-level qualifier throughout the JCL reflects your installation.

Developing the Application Interfaces

Overview

This section describes how to develop the interfaces to the objects that are to be implemented in the FNB server. It first describes the IDL interfaces on which the FNB objects are based. It then describes how to generate COBOL source and copybooks from these IDL interfaces, and provides a description of the various members generated.

In this section

This section discusses the following topics:

Defining IDL Interfaces

Overview

The first step in writing any Orbix application is to define the IDL interfaces for the objects required in your system. This section provides a very brief overview of IDL and its advantages before then showing and describing the IDL definitions for the objects implemented by the FNB COBOL back-end server. It discusses the following topics:

- "OMG IDL" on page 22.
- "Data definitions IDL" on page 22.
- "FNB IDL" on page 24.

Note: The IDL interfaces are already supplied for you, in the `*orbixhlq*.DEMOS.IDL` PDS, so this subsection is provided for the purposes of illustration.

OMG IDL

The OMG interface definition language (IDL) is a purely declarative language, with a syntax similar to C++ and Java, that is used to define the interfaces for CORBA objects.

The advantage of OMG IDL is that it enables you to define distributed interfaces in a language-neutral manner.

A server developer can use IDL to define the service provided to clients, irrespective of the language or platform used on the server side. Conversely, a client programmer can use IDL as a blueprint for accessing the service, irrespective of the language or platform used on the client side.

For more details about IDL in general see the *COBOL Programmer's Guide and Reference*.

Data definitions IDL

Example 1 shows the data definitions IDL member. This IDL member is contained in `*orbixhlq*.DEMOS.IDL(DATADEFS)` and defines some basic data types used in other parts of the IDL.

Example 1: *Data Definitions IDL Member*

```
   // IDL
1  #ifndef DATADEFS_IDL
   #define DATADEFS_IDL
```

Example 1: *Data Definitions IDL Member*

```
2  typedef long accountNum;
3  typedef sequence<accountNum> accountNumList;

   #endif // DATADEFS_IDL
```

The preceding code can be explained as follows:

1. An IDL member can contain preprocessor macros, similar to the C and C++ languages. The start of a macro is signalled by a # character at the beginning of a line.

   In this example, the `#ifndef`, `#define`, and `#endif` preprocessor macros guard against multiple inclusion of this file into other IDL members.

2. The `typedef` construction is grammatically similar to `typedef` in C and C++. In this example, `accountNum` becomes a synonym for the IDL `long` type (32-bit signed integer).

3. This line defines a *sequence type*, `accountNumList`, defined as an unbounded sequence of integers, `accountNum`. A sequence is similar to a one-dimensional array except that its length can be arbitrary.

   For example, the IDL-to-COBOL mapping specifies that for the purposes of mapping an IDL unbounded sequence to COBOL, a group item is created to hold one element of the sequence, and a supporting group item is also created. The supporting group item contains data definitions that define the maximum number of elements for the sequence, the number of elements currently populated in the sequence, the actual data associated with each element, and the typecode associated with the sequence.

   Because the elements of a sequence are not directly accessible, you can call `SEQSET` to copy the supplied data into the requested element of the sequence, and `SEQGET` to provide access to a specific element of the sequence. Because an unbounded sequence is a dynamic type, memory must be allocated for it at runtime, by calling `SEQALLOC`.

Note: See the *COBOL Programmer's Guide and Reference* for more details of IDL-to-COBOL mapping rules, `SEQGET`, `SEQSET`, and `SEQALLOC`.

FNB IDL

Example 2 shows the main IDL member used by the FNB COBOL back-end server. This IDL member is contained in *`orbixhlq`*`.DEMOS.IDL(FNB)` and defines all the CORBA interfaces implemented by the back-end server.

Example 2: *FNB IDL Member (Sheet 1 of 3)*

```
   // IDL
   #ifndef FNB_IDL
   #define FNB_IDL
1  #include "DATADEFS"

   // Exceptions raised in this file

2  module bankobjects {
3      exception INSUFFICIENT_FUNDS {};
       exception CANNOT_CLOSE_ACCOUNT {};
       exception ACCOUNT_DOESNT_EXIST {};
       exception FAILED_TO_AUTHORIZE {};

4      struct address {
           string address_1;
           string address_2;
           string address_3;
       };

       // Stucture to hold information on what a customer
       // is doing with the bank
       struct BankTransaction {
           short id;
           string date;
           string record_type;
           string value;
       };

5      typedef sequence<BankTransaction> AccountTransactions;
6      interface Account;
```

Example 2: *FNB IDL Member (Sheet 2 of 3)*

```
7       interface AccountMgr  {
8           Account openAccount ( in accountNum accountNumber)
                raises (ACCOUNT_DOESNT_EXIST);
            Account newAccount (in string accountType);
            void closeAccount (in accountNum accountNumber )
                raises (CANNOT_CLOSE_ACCOUNT);

            accountNumList getCurrentAccountList ();
            accountNumList getCreditCardList ();
        };

        interface Account {
9           readonly attribute accountNum accountnumber;
            readonly attribute address addr;
            readonly attribute string accountType;

10          attribute string firstname;
            attribute string lastname;

            readonly attribute float accountBalance;

            readonly attribute AccountTransactions
               recentTransactions;

            // Update methods
            boolean makeLodgement (in float amount );
            boolean withdrawFunds (in float amount)
              raises (INSUFFICIENT_FUNDS);
            boolean updateAddress (in address newAddress);

            void transferFundsIn (in float amount );
            void transferFundsOut (in float amount )
              raises (INSUFFICIENT_FUNDS);

            // Admin stuff
            void sendStatement ();
        };

11      interface CurrentAccount : Account {
            readonly attribute float overdraftLimit;

            // Account maintenace routines
            boolean approveNewOverdraft (in float amount);
        };
```

Example 2: *FNB IDL Member (Sheet 3 of 3)*

```
    interface SavingsAccount : Account {
    };

    typedef short authorizationCode;

    interface CreditCardAccount : Account {
        attribute float limit;
        attribute float interest_rate;

        // Calculate how much interest is owed on this account
        float calculateInterest ();

        // Basic operations on a credit card
        authorizationCode authoriseAmount (in float amount)
          raises (FAILED_TO_AUTHORIZE);
        boolean makePurchase (in string vendor, in float amount,
                              in authorizationCode auth_code);
    };

}; // Module
#endif //ACCOUNT_IDL
```

The preceding code can be explained as follows:

1. Definitions from the `DATADEFS` IDL member (see "Data definitions IDL" on page 22) are included in this file by calling the `#include` preprocessor macro.
2. The definitions in the `FNB` IDL member are enclosed within the `bankobjects` module. An IDL module is a scoping mechanism for IDL.

   All the entities defined in the scope of the `bankobjects` module gain `bankobjects::` as a prefix. For example, `bankobjects::Account` is the fully scoped identifier for the `Account` interface.
3. This line and the following lines define some IDL *user exception* types. The exception definitions shown here have an empty body, `{}`, because there is no data associated with these user exceptions.
4. The syntax for declaring an IDL `struct` is similar to the syntax of a C++ `struct`.

   For example, the `address` struct type contains three strings corresponding to the three fields of an address, `address_1`, `address_2`, and `address_3`.

5. The `typedef` declares an unbounded sequence, `AccountTransactions`, that holds a list of `BankTransaction` structs. A sequence should always be declared using a `typedef` construction.
6. This is an example of a forward declaration of an interface, `Account`. This enables the `Account` interface to be referenced before it is defined. The actual definition of the `Account` interface appears further on.
7. This line introduces the definition of an IDL interface, `AccountMgr`. Interfaces are the most important sort of definition in IDL. An IDL interface defines the attributes and operations for CORBA objects of a particular type.
8. This line shows an example of an *IDL operation*, `openAccount()`. A `raises()` clause introduces the list of user exceptions that can be thrown by this operation.
9. A readonly attribute in an IDL interface maps to an operation with a `-GET-` prefix that enables you to retrieve the attribute value..

   Note: See the *COBOL Programmer's Guide and Reference* for more details of IDL-to-COBOL mapping rules.

10. An attribute that is not readonly maps to two operations: one with a `-GET-` prefix that enables you to retrieve the attribute value, and one with a `-SET-` prefix that enables you to update the attribute value.
11. The `CurrentAccount` interface inherits from `Account`. IDL inheritance is indicated using `:` (that is, a colon). Multiple inheritance is supported in IDL.

Orbix IDL Compiler

Overview

This subsection describes how to configure and run the Orbix IDL compiler to generate COBOL source and copybooks from IDL definitions. It discusses the following topics:

- "Using Orbix IDL compiler" on page 28.
- "Orbix IDL compiler configuration" on page 28.
- "Configuration settings" on page 29.
- "Configuration settings explanation" on page 29.
- "Generating COBOL copybooks for Naming Service" on page 32.

Note: See the *COBOL Programmer's Guide and Reference* for more details of the Orbix IDL compiler, including all the arguments that you can use with it.

Using Orbix IDL compiler

To access the definitions expressed in IDL, it is necessary to compile the IDL into a target language such as COBOL. This is accomplished using the *IDL compiler*, which takes an IDL file as input and generates server skeleton files as output.

Note: For certain languages, such as C++ or Java, the Orbix IDL compiler generates client stub files as well as server skeleton files. It does not, however, generate client stub files for COBOL.

Orbix IDL compiler configuration

The Orbix IDL compiler uses the Orbix configuration member for its settings. The `FNBIDL` JCL that runs the Orbix IDL compiler on OS/390 uses a configuration member provided in `*orbixhlq*.CONFIG(IDL)`.

Configuration settings

The COBOL configuration for the Orbix IDL compiler is listed under `Cobol` as follows:

```
Cobol
{
    Switch = "cobol";
    ShlibName = "ORXBCBL";
    ShlibMajorVersion = "x";
    IsDefault = "NO";
    PresetOptions = "";
# COBOL source and copybooks extensions
# The default is .cbl, .xxx and .cpy on NT and none for OS/390.
    CobolExtension = "";
    ImplementationExtension = "";
    CopybookExtension = "";
};
```

Note: Settings listed with a # are considered to be comments and are not in effect. The default in relation to COBOL source and copybooks extensions is also none for OS/390 UNIX System Services.

Configuration settings explanation

The available configuration settings can be explained as follows:

Table 3: *COBOL Configuration Settings (Sheet 1 of 3)*

Variable Name	Description	Default
`Switch`	This informs the Orbix IDL compiler how to recognise the COBOL switch that indicates to generate COBOL code. This setting is mandatory and must not be altered.	
`ShlibName`	This informs the Orbix IDL compiler what name the DLL plug-in is stored under. This setting is mandatory and must not be altered.	

Table 3: *COBOL Configuration Settings (Sheet 2 of 3)*

Variable Name	Description	Default
`ShlibMajorVersion`	This is the version number of the supplied ShlibName DLL. This setting is mandatory and must not be altered.	
`IsDefault`	Indicates whether COBOL is the language that the Orbix IDL compiler generates by default from IDL. If this is set to `YES`, you do not need to specify the `-cobol` switch when running the compiler.	
`PresetOptions`	The arguments that are passed by default as parameters to the Orbix IDL compiler.	
`CobolExtension`[a]	Extension for the server mainline source code file on OS/390 UNIX System Services and Windows NT. **Note:** This is left blank by default, and you can set it to any value you want. The recommended setting is `.cbl`.	
`ImplementationExtensiona`	Extension for the server implementation source code filename on OS/390 UNIX System Services and Windows NT. You should copy this to a file with a `.cbl` extension, to avoid overwriting any subsequent changes if you run the Orbix IDL compiler again. **Note:** This is left blank by default, and you can set it to any value you want. The recommended setting is `.xxx`.	

Table 3: *COBOL Configuration Settings (Sheet 3 of 3)*

Variable Name	Description	Default
`CopybookExtension`[a]	Extension for COBOL copybook names on OS/390 UNIX System Services and Windows NT. **Note:** This is left blank by default, and you can set it to any value you want. The recommended setting is `.cpy`.	
`MainCopybookSuffix`	Suffix for the main copybook member name.	
`RuntimeCopybookSuffix`	Suffix for the runtime copybook member name.	`X`
`SelectCopybookSuffix`	Suffix for the select copybook member name.	`D`
`ImplementationSuffix`	Suffix for the server implementation source code member name.	`S`
`ServerSuffix`	Suffix for the server mainline source code member name.	`SV`

a. This is ignored on native OS/390.

Note: The last five variables in Table 3 are not listed by default in *`orbixhlq`*`.CONFIG(IDL)`. If you want to change the generated member suffixes from the default values shown in Table 3, you must manually enter the relevant variable name and its corresponding value.

Generating alternative mapping entries

The Orbix IDL compiler can take various arguments as parameters. See the *COBOL Programmer's Guide and Reference* for full details of these. One of these arguments, `-M`, allows you to set up an alternative mapping scheme for data names. By default, the Orbix IDL compiler generates COBOL data names based on fully scoped interface names. This can lead to unwieldy and possibly truncated identifier names.

To allow you to specify an alternative and more meaningful naming scheme for your COBOL identifiers, you can specify the `-M` argument when you run the IDL compiler, to generate a mapping member that contains a more logical naming scheme. The following is an example of the contents of the supplied mapping member for the FNB demonstration:

```
bankobjects bo
bankobjects/Account Account
bankobjects/AccountMgr AccMgr
bankobjects/CurrentAccount CA
bankobjects/SavingsAccount SA
bankobjects/CreditCardAccount CCA
bankobjects/CreditCardAccount/limit CCA-LIMIT
bankobjects/BankTransaction/id TXN-ID
bankobjects/BankTransaction/date TXN-DATE
bankobjects/BankTransaction/record_type TXN-RECORD-TYPE
bankobjects/BankTransaction/value TXN-VALUE
```

For example, based on the preceding mapping member example, the alternative name for the `bankobjects/CreditCardAccount` identifier is `CCA`.

Generating COBOL copybooks for Naming Service

Before you run the Orbix IDL compiler to generate the COBOL copybooks for the FNB demonstration server, run the Orbix IDL compiler to generate the COBOL copybooks for the Naming Service. To do this, submit *orbixhlq*`.DEMOS.COBOL.BLD.JCL(NAMESIDL)`. This takes as input the IDL defined in *orbixhlq*`.INCLUDE.OMG.IDL(COSNAMI)` for the Naming Service and generates the COBOL copybooks `NAMES`, `NAMESX`, and `NAMESD` in *orbixhlq*`.DEMOS.COBOL.COPYLIB`.

In this case, the `NAMESIDL` JCL specifies the `-O` argument with the Orbix IDL compiler, to generate alternative copybook names instead of allowing the generated copybook names to be automatically based on the IDL member name, `COSNAMI`.

Generating COBOL copybooks for the FNB server

Submit *orbixhlq*`.DEMOS.COBOL.BLD.JCL(FNBIDL)` to run the Orbix IDL compiler, to generate the COBOL copybooks for the FNB server. This takes as input the IDL defined in *orbixhlq*`.DEMOS.IDL(FNB)` for the FNB demonstration and generates the COBOL copybooks `FNB`, `FNBX`, and `FNBD` in *orbixhlq*`.DEMOS.COBOL.COPYLIB`.

The source code members for the FNB COBOL back-end server are already generated and shipped with Orbix Mainframe. The arguments to generate the relevant source code members are therefore disabled in `orbixhlq.DEMOS.COBOL.BLD.JCL(FNBIDL)`. See the *COBOL Programmer's Guide and Reference* for full details of IDL compiler arguments.

Generated Source Code and Copybooks

Overview

This subsection provides an overview of the source code and copybook members that the Orbix IDL compiler generates for the FNB COBOL back-end server.

Generated source code members

Table 4 provides an overview of the server source code members that the Orbix IDL compiler generates, based on the defined IDL.

Note: These are already generated for you for the purposes of this demonstration. They are provided in the *orbixhlq*.DEMOS.COBOL.SRC PDS.

Table 4: *Generated FNB Server Source Code Members*

Source Member Name	JCL Keyword Parameter	Description
FNBS	IMPL	This is the server implementation source code member. It contains stub paragraphs for all the callable operations.
FNBSV	IMPL	This is the server mainline source code member.

Generated copybooks

Table 5 provides an overview of the COBOL copybook members that the Orbix IDL compiler generates, based on the defined IDL, when you submit the *orbixhlq*.DEMOS.COBOL.BLD.JCL(FNBIDL) JCL.

Table 5: *Generated COBOL Copybooks*

Copybook	JCL Keyword Parameter	Description
FNB	COPYLIB	The FNB copybook contains data definitions that are used for working with operation parameters and return values for each interface defined in the FNB IDL member.
FNBX	COPYLIB	The FNBX copybook contains data definitions that are used by the COBOL runtime to support the interfaces defined in the FNB IDL member. This copybook is automatically included in the FNB copybook.
FNBD	COPYLIB	The FNBD copybook contains procedural code for performing the correct paragraph for the requested operation. This copybook is automatically included in the FNBS server implementation source code member.

How IDL maps to COBOL copybooks

Each IDL interface maps to a group of COBOL data definitions. There is one definition for each IDL operation. A definition contains each of the parameters for the relevant IDL operation in their corresponding COBOL representation. See the COBOL Programmer's Guide and Reference for details of how IDL types map to COBOL.

Attributes map to two operations (get and set), and readonly attributes map to a single get operation.

Location of demonstration source

You can find examples of the source code for the `FNB` back-end server demonstration in the following locations:

- *`orbixhlq`*`.DEMOS.COBOL.SRC(FNBSV)`
- *`orbixhlq`*`.DEMOS.COBOL.SRC(FNBS)`

Note: These source code members are shipped with your Orbix Mainframe installation.

Location of demonstration copybooks

You can find examples of the copybooks generated for the `FNB` back-end server demonstration in the following locations:

- *`orbixhlq`*`.DEMOS.COBOL.COPYLIB(FNB)`
- *`orbixhlq`*`.DEMOS.COBOL.COPYLIB(FNBX)`
- *`orbixhlq`*`.DEMOS.COBOL.COPYLIB(FNBD)`

Note: These copybooks are not shipped with your Orbix Mainframe installation. They are generated when you run the supplied JCL in *`orbixhlq`*`.DEMOS.COBOL.BLD.JCL(FNBIDL)`, to run the Orbix IDL compiler.

Writing the Server

Overview

This section describes the steps you must follow to develop the server executable for the FNB back-end server demonstration.

Note: This section is provided for the purposes of illustration only. The server is supplied fully written with your Orbix Mainframe installation.

Steps to develop the server

The steps to develop the server application are:

Step	Action
1	"Writing the Server Implementation" on page 38
2	"Writing the Server Mainline" on page 62

Writing the Server Implementation

The server implementation module

You must implement the server interface by writing a COBOL module that implements each operation in the generated `FNB` copybook.

Note: Ordinarily, when you specify the `-Z` argument with the Orbix IDL compiler, it generates a module called *`idlmembername`*`S`, which contains the server skeleton implementation code. For the purposes of this demonstration, however, the `FNBS` module is provided fully implemented for you. The `-Z` argument is therefore disabled by default in the `FNBIDL` JCL that you use to run the IDL compiler for this demonstration.

Example of the FNBS module

Example 3 shows parts of the `FNBS` module (ellipses are used to denote code omitted for the sake of brevity):

Note: You can find the complete `FNBS` server implementation program in *`orbixhlq`*`.DEMOS.COBOL.SRC(FNBS)`.

Example 3: *FNBS Server Implementation Module (Sheet 1 of 11)*

```
****************************************************************
*   Copyright (c) 2001-2003 IONA Technologies PLC.
*   All Rights Reserved.
*
* Description: This is the  batch server implementation of the
*              FNB demo.
*
****************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID.             FNBS.

ENVIRONMENT DIVISION.

1 INPUT-OUTPUT SECTION.
…

DATA DIVISION.

2 FILE SECTION.
…
```

Example 3: *FNBS Server Implementation Module (Sheet 2 of 11)*

```
    WORKING-STORAGE SECTION.
    …

3   COPY FNB.
4   COPY CORBA.


5   LINKAGE SECTION.
    01 LS-ACCOUNT-CHAIN-ENTRY.
       05 LS-ACCOUNT-IOR                          POINTER.
       05 LS-ACCOUNT-NUMBER                       PIC 9(10) BINARY.
       05 LS-ACCOUNT-NEXT-ENTRY                   POINTER.

    PROCEDURE DIVISION.
6   ENTRY "DISPATCH".

7   CALL "COAREQ" USING REQUEST-INFO.
    SET WS-COAREQ TO TRUE.
    PERFORM CHECK-STATUS.
    * Resolve the pointer reference to the interface name which is
    * the fully scoped interface name
8   CALL "STRGET" USING INTERFACE-NAME
                             WS-INTERFACE-NAME-LENGTH
                             WS-INTERFACE-NAME.
                             SET WS-STRGET TO TRUE.
                             PERFORM CHECK-STATUS.

    **************************************************************
    * Interface(s) :
    **************************************************************
    MOVE SPACES TO ACCMGR-OPERATION.
    MOVE SPACES TO ACCOUNT-OPERATION.
    MOVE SPACES TO CA-OPERATION.
    MOVE SPACES TO CCA-OPERATION.

    ***************************************************************
    * Evaluate Interface(s) :
    ***************************************************************

    EVALUATE WS-INTERFACE-NAME
    WHEN 'IDL:bankobjects/AccountMgr:1.0'

    * Resolve the pointer reference to the operation information
9   CALL "STRGET" USING OPERATION-NAME
```

Example 3: *FNBS Server Implementation Module (Sheet 3 of 11)*

```
                    ACCMGR-OPERATION-LENGTH
                    ACCMGR-OPERATION
SET WS-STRGET TO TRUE
PERFORM CHECK-STATUS
WHEN 'IDL:bankobjects/Account:1.0'

* Resolve the pointer reference to the operation information
CALL "STRGET" USING OPERATION-NAME
                    ACCOUNT-OPERATION-LENGTH
                    ACCOUNT-OPERATION
SET WS-STRGET TO TRUE
PERFORM CHECK-STATUS
WHEN 'IDL:bankobjects/CurrentAccount:1.0'

* Resolve the pointer reference to the operation information
CALL "STRGET" USING OPERATION-NAME
                    CA-OPERATION-LENGTH
                    CA-OPERATION
SET WS-STRGET TO TRUE
PERFORM CHECK-STATUS
WHEN 'IDL:bankobjects/CreditCardAccount:1.0'

* Resolve the pointer reference to the operation information
CALL "STRGET" USING OPERATION-NAME
                    CCA-OPERATION-LENGTH
                    CCA-OPERATION
SET WS-STRGET TO TRUE
PERFORM CHECK-STATUS
END-EVALUATE.

COPY FNBD.
GOBACK.
ENTRY "SHUTDOWN".

* This entry point is not really needed but is used to demon
* -strate how dynamic storage can be freed again. The dynamic
* storage is in this case the linked list containing the account
* object references that have been already been created. This
* entry point is called at the end of the server main program
* when it is shutting down.


SET ADDRESS OF LS-ACCOUNT-CHAIN-ENTRY TO
               WS-ACCOUNT-ANCHOR
PERFORM UNTIL ADDRESS OF LS-ACCOUNT-CHAIN-ENTRY = NULL
```

Margin callouts: 9 (beside first CALL "STRGET" ... ACCOUNT-OPERATION), 9 (beside CALL "STRGET" ... CA-OPERATION), 9 (beside CALL "STRGET" ... CCA-OPERATION), 10 (beside COPY FNBD.)

Example 3: *FNBS Server Implementation Module (Sheet 4 of 11)*

```
        CALL "OBJREL" USING LS-ACCOUNT-IOR
        SET WS-OBJREL TO TRUE
        PERFORM CHECK-STATUS

        SET WS-ACCOUNT-ENTRY-PTR TO
            ADDRESS OF LS-ACCOUNT-CHAIN-ENTRY
        SET ADDRESS OF LS-ACCOUNT-CHAIN-ENTRY TO
            LS-ACCOUNT-NEXT-ENTRY

        CALL "MEMFREE" USING WS-ACCOUNT-ENTRY-PTR
        SET WS-MEMFREE TO TRUE
        PERFORM CHECK-STATUS

    END-PERFORM.
    GOBACK.

12  DO-ACCMGR-OPENACCOUNT.
    SET D-NO-USEREXCEPTION TO TRUE.
11  CALL "COAGET" USING ACCMGR-OPENACCOUNT-ARGS.
    SET WS-COAGET TO TRUE.
    PERFORM CHECK-STATUS.

    MOVE ACCOUNTNUMBER OF ACCMGR-OPENACCOUNT-ARGS TO
                          ACCOUNT-NUMBER.

    READ ACCOUNTS KEY IS ACCOUNT-KEY
    END-READ.

    IF ( ACCOUNT-STATUS NOT = 0
           AND ACCOUNT-STATUS NOT = 2
                AND ACCOUNT-STATUS NOT = 23 )
        DISPLAY '*** A response of, ' ACCOUNT-STATUS ', was '
                '`detected when reading the Account file.'
        GOBACK
    END-IF

    IF ACCOUNT-STATUS = 23
        DISPLAY " account number not found "
        SET D-BO-ACCOUNT-DOESNT-EXIST TO TRUE

        DISPLAY "set exception id ..."
        CALL "STRSET" USING EXCEPTION-ID
                            EX-BO-ACCOUNT-DOESNT-EXIS-4410
                            EX-BO-ACCOUNT-DOESNT-EXIST
```

Example 3: *FNBS Server Implementation Module (Sheet 5 of 11)*

```
    SET WS-STRSET TO TRUE
    PERFORM CHECK-STATUS

    DISPLAY "raise user exception with coaerr..."
    CALL "COAERR" USING FNB-USER-EXCEPTIONS

    SET WS-COAERR TO TRUE
    PERFORM CHECK-STATUS

END-IF
* the account exists in the datastore. Have we created an
* object reference for it? Search the linked list first.
* These searches are extremely fast despite being O(N).
* If the search fails then call objnew to create the object
* and insert it into the linked list.
PERFORM SEARCH-ACCOUNT-CHAIN
IF WS-QUIT-SEARCH-ACCOUNT-LOOP = 0 THEN
    DISPLAY " no record in linked list for "
    DISPLAY " account number " ACCOUNT-NUMBER
    DISPLAY " call objnew "
* set up the call to OBJNEW.
EVALUATE ACCOUNT-CLASS
    WHEN 'Credit Card'
        MOVE "IDL:bankobjects/CreditCardAccount:1.0 "
            TO  WS-INTERFACE-NAME
    WHEN 'Current'
        MOVE "IDL:bankobjects/CurrentAccount:1.0 "
            TO  WS-INTERFACE-NAME
END-EVALUATE

* convert binary account number key to string - must
* be at least one trailing space for use in OBJNEW
* call
    MOVE ACCOUNT-NUMBER TO WS-ACCOUNT-NUMBER-STR11
    MOVE SPACES TO WS-ACCOUNT-NUMBER-STR11(11:1)

* create new account object reference
    CALL "OBJNEW" USING SERVER-NAME
                        WS-INTERFACE-NAME
                        WS-ACCOUNT-NUMBER-STR11
                        WS-OBJ-COPY
    SET WS-OBJNEW TO TRUE
    PERFORM CHECK-STATUS
* OBJNEW was Ok - add the new obj reference to the linked list
    MOVE LENGTH OF LS-ACCOUNT-CHAIN-ENTRY
```

Example 3: *FNBS Server Implementation Module (Sheet 6 of 11)*

```
                   TO WS-TEMP-LENGTH
    CALL "MEMALLOC" USING WS-TEMP-LENGTH
                          WS-ACCOUNT-ENTRY-PTR
    SET WS-MEMALLOC TO TRUE
    PERFORM CHECK-STATUS

* set current chain entry to newly allocated memory
    SET ADDRESS OF LS-ACCOUNT-CHAIN-ENTRY TO
                       WS-ACCOUNT-ENTRY-PTR
* write the newly created account number to the chain entry
    MOVE ACCOUNT-NUMBER TO LS-ACCOUNT-NUMBER
* duplicate object just created in order to prevent deletion
* when COAPUT runs
    SET LS-ACCOUNT-IOR TO WS-OBJ-COPY
* finally insert the new chain entry at the head of the chain.
    SET LS-ACCOUNT-NEXT-ENTRY TO WS-ACCOUNT-ANCHOR
    SET WS-ACCOUNT-ANCHOR TO WS-ACCOUNT-ENTRY-PTR
    CALL "OBJDUP" USING LS-ACCOUNT-IOR
                        RESULT OF
                        ACCMGR-OPENACCOUNT-ARGS
    SET WS-OBJDUP TO TRUE
    PERFORM CHECK-STATUS
ELSE
    CALL "OBJDUP" USING LS-ACCOUNT-IOR
                        RESULT OF
                        ACCMGR-OPENACCOUNT-ARGS
    SET WS-OBJDUP TO TRUE
    PERFORM CHECK-STATUS
*  end of linked list search test ********************
END-IF.

EVALUATE TRUE
WHEN D-NO-USEREXCEPTION
```
11
```
CALL "COAPUT" USING ACCMGR-OPENACCOUNT-ARGS
SET WS-COAPUT TO TRUE
PERFORM CHECK-STATUS
END-EVALUATE.
```
13
```
DO-ACCMGR-NEWACCOUNT.
…
```
14
```
DO-ACCMGR-CLOSEACCOUNT.
…
```
15
```
DO-ACCMGR-GETCURRENTACCOU-AE9B.
```

Example 3: *FNBS Server Implementation Module (Sheet 7 of 11)*

```
   …
15 DO-ACCMGR-GETCREDITCARDLIST.
   …
16 DO-ACCOUNT-GET-ACCOUNTNUMBER.
   …
17 DO-ACCOUNT-GET-ADDR.
   …
18 DO-ACCOUNT-GET-ACCOUNTTYPE.
   …
18 DO-ACCOUNT-GET-FIRSTNAME.
   …
19 DO-ACCOUNT-SET-FIRSTNAME.
   …
18 DO-ACCOUNT-GET-LASTNAME.
   …
19 DO-ACCOUNT-SET-LASTNAME.
   …
20 DO-ACCOUNT-GET-ACCOUNTBALANCE.
   …
16 DO-ACCOUNT-GET-RECENTTRAN-D044.
   …
21 DO-ACCOUNT-MAKELODGEMENT.
   …
22 DO-ACCOUNT-UPDATEADDRESS.
   …
21 DO-ACCOUNT-TRANSFERFUNDSIN.
   …
23 DO-CA-GET-OVERDRAFTLIMIT.
   …
24 DO-CA-APPROVENEWOVERDRAFT.
```

Example 3: *FNBS Server Implementation Module (Sheet 8 of 11)*

```
   …
25 DO-CA-GET-ACCOUNTNUMBER.
   …
26 DO-CA-GET-ADDR.
   …
27 DO-CA-GET-ACCOUNTTYPE.
   …
28 DO-CA-GET-FIRSTNAME.
   …
29 DO-CA-SET-FIRSTNAME.
   …
30 DO-CA-GET-LASTNAME.
   …
31 DO-CA-SET-LASTNAME.
   …
32 DO-CA-GET-ACCOUNTBALANCE.
   …
33 DO-CA-GET-RECENTTRANSACTIONS.
   …
34 DO-CA-MAKELODGEMENT.
   …
35 DO-CA-WITHDRAWFUNDS.
   …
36 DO-CA-UPDATEADDRESS.
   …
37 DO-CA-TRANSFERFUNDSIN.
   …
38 DO-CA-TRANSFERFUNDSOUT.
   …
39 DO-CCA-GET-LIMIT.
```

Example 3: *FNBS Server Implementation Module (Sheet 9 of 11)*

```
   …
40 DO-CCA-SET-LIMIT.
   …
39 DO-CCA-GET-INTEREST-RATE.
   …
40 DO-CCA-SET-INTEREST-RATE.
   …
41 DO-CCA-CALCULATEINTEREST.
   …
42 DO-CCA-AUTHORISEAMOUNT.
   …
43 DO-CCA-MAKEPURCHASE.
   …
25 DO-CCA-GET-ACCOUNTNUMBER.
   …
26 DO-CCA-GET-ADDR.
   …
27 DO-CCA-GET-ACCOUNTTYPE.
   …
28 DO-CCA-GET-FIRSTNAME.
   …
29 DO-CCA-SET-FIRSTNAME.
   …
30 DO-CCA-GET-LASTNAME.
   …
31 DO-CCA-SET-LASTNAME.
   …
32 DO-CCA-GET-ACCOUNTBALANCE.
   …
33 DO-CCA-GET-RECENTTRANSACTIONS.
```

Example 3: *FNBS Server Implementation Module (Sheet 10 of 11)*

```
   …

34 DO-CCA-MAKELODGEMENT.
   …

35 DO-CCA-WITHDRAWFUNDS.
   …

36 DO-CCA-UPDATEADDRESS.
   …

37 DO-CCA-TRANSFERFUNDSIN.
   …

38 DO-CCA-TRANSFERFUNDSOUT.
   …


   ****************************************************************
   * Check Errors Copybook
   ****************************************************************
       *
44 COPY CHKERRS.
   FIND-LAST-ACCNUM.
   *=================
   …

   UPDATE-LAST-ACCNUM.
   *==================
   …

   CREATE-NEW-ACCOUNT.
   *===================
   …

   GET-OBJECTID-FROM-TARGET.
   *=========================
   …

   RETRIEVE-ACCOUNT-DETAILS.
   *=========================
   …

   UPDATE-ACCOUNT-DETAILS.
   *=========================
```

Example 3: *FNBS Server Implementation Module (Sheet 11 of 11)*

```
…

BUILD-ACCOUNT-SEQUENCE.
*=========================
…

BUILD-CCA-SEQUENCE.
*===================
…

BUILD-TXNHIST-SEQUENCE.
*=========================
…

BUILD-CCA-TXNHIST-SEQUENCE.
*===========================
…

SEARCH-ACCOUNT-CHAIN.
*=====================
…

GET-TXN-ID.
*===========
…

UPDATE-TXNNUM.
*==============
…

UPDATE-TXNHIST.
*==============
…

CREATE-TXN-HIST.
*================
…
```

Explanation of the batch FNBS module

The FNBS module can be explained as follows:

1. This section defines the files to be used by the server application for storing account data (ACCOUNTS), last-account-number-used data (ACCNUM), transaction history data (TXNHIST), and last-transaction-history-key-used-per-account data (TXNNUM).
2. This section defines the layout of the records in each of the files used by the server application.
3. The data definitions used for working with operation parameters and return values for each interface being implemented are copied in from the FNB copybook.
4. Various Orbix COBOL definitions, such as REQUEST-INFO used by the COAREQ function, and ORBIX-STATUS-INFORMATION used to register and report system exceptions raised by the COBOL runtime, are copied in from the CORBA copybook.
5. This section defines the layout of the data in the linked list for recording currently active objects.
6. The DISPATCH logic is automatically coded for you, and the bulk of the code is contained in the FNBD copybook. When an incoming request arrives from the network, it is processed by the ORB and a call is made to the DISPATCH entry point.
7. COAREQ is called to provide information about the current invocation request, which is held in the REQUEST-INFO block that is contained in the CORBA copybook.

   COAREQ is called once for each operation invocation—after a request has been dispatched to the server, but before any calls are made to access the parameter values.
8. STRGET is called to copy the characters in the unbounded string pointer for the interface name to the string item representing the fully scoped interface name.
9. STRGET is called again to copy the characters in the unbounded string pointer for the relevant operation name, in each interface respectively, to the string item representing the operation name.

10. The procedural code used to perform the correct paragraph for the requested operation is copied into the program from the `FNBD` copybook.
11. This operation (and every other operation) calls `COAGET` and `COAPUT` to copy incoming values and return values, respectively, from and to the COBOL structures for the operation's parameter list. `COAGET` and `COAPUT` must be called by every operation, even if the operation takes no parameters or returns no values.
12. The `DO-ACCMGR-OPENACCOUNT` operation:
    1. Moves the account number passed in from the client to file.
    2. Reads the account file, to check if the account already exists, using the account number as the key.
    3. Checks the account status to see if the account exists on file.
    4. If the account does not exist, the server calls `COAERR`, to raise the `ACCOUNT_DOESNT_EXIST` user exception.
    5. If the account does exist, the server searches the linked list, to see if an object reference already exists for it.

       If an object reference does not exist for the account, the server checks the `ACCOUNT-CLASS` value on the account record to see if it is a credit card or current account, calls `OBJNEW` to create an object reference, calls `MEMALLOC` to dynamically allocate memory in the account chain for the account's IOR, and then moves the account number to the newly created entry in the account chain.

       Alternatively, if an object reference already exists, the server calls `OBJDUP` to create a copy of the object reference, to conform with the memory management rules for object references (see the *COBOL Programmer's Guide and Reference* for more details).
13. The `DO-ACCMGR-NEWACCOUNT` operation:
    1. Performs the `FIND-LAST-ACCNUM` paragraph, to find out the last account number created in the `ACCNUM` data set.
    2. Initializes an account record and assigns it an account number equal to the last account number plus one.
    3. Calls `STRGET`, using the account type passed in, to see what type of account is to be created.

- iv. Converts the binary account number key to a string.
- v. Calls `MEMALLOC` to dynamically allocate memory in the account chain for the new account's IOR.
- vi. Calls `OBJNEW` to create an object reference for the new account, and then moves the newly created account number to the newly created entry in the account chain.
- vii. Calls `OBJDUP` to create a copy of the object reference, to conform with the memory management rules for object references (see the *COBOL Programmer's Guide and Reference* for more details)..
- viii. Performs the `CREATE-NEW-ACCOUNT` paragraph to create the new account record in the `ACCOUNTS` data set, and then performs the `UPDATE-LAST-ACCNUM` paragraph to store the new account number as the last account number created in the `ACCNUM` data set.
- ix. Finally, it performs the `CREATE-TXN-HIST` paragraph, to update the transaction history for the new account in the `TXNHIST` data set.

14. The `DO-ACCMGR-CLOSEACCOUNT` operation:
    - i. Moves the account number passed in to the record key of the accounts file (`ACCOUNTS`).
    - ii. If the record exists on file, it is deleted. Otherwise, the server raises the `CANNOT_CLOSE_ACCOUNT` user exception.
15. The `DO-ACCMGR-GETCURRENTACCOU-AE9B` and `DO-ACCMGR-GETCREDITCARDLIST` operations:
    - i. Moves the literal value relating to the account type (that is `"Current"` or `"Credit Card"`) to file.
    - ii. Reads the account file, using the account class (that is, the account type) as the key.
    - iii. Calls `SEQALLOC`, if no accounts of that type exist, to return a sequence of zero length.
    - iv. Calls `SEQALLOC`, if accounts of that type do exist, to return a sequence of those accounts.

16. The `DO-ACCOUNT-GET-ACCOUNTNUMBER` and `DO-ACCOUNT-GET-RECENTTRAN-D044` generic operations:
    i. Perform the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    ii. Convert the account number in Working Storage to a numeric string.
17. The `DO-ACCOUNT-GET-ADDR` generic operation:
    i. Performs the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    ii. Converts the account number in Working Storage to a numeric string.
    iii. Performs the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file, based on the account number, to retrieve the account details.
    iv. Moves the account record address line 1 and its length to Working Storage.
    v. Calls `STRSET` to create an unbounded string from address line 1 in Working Storage.
    vi. Repeats steps v and vi for address line 2 and address line 3.
18. The `DO-ACCOUNT-GET-ACCOUNTTYPE`, `DO-ACCOUNT-GET-FIRSTNAME`, and `DO-ACCOUNT-GET-LASTNAME` generic operations:
    i. Perform the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    ii. Convert the account number in Working Storage to a numeric string.
    iii. Perform the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file, based on the account number, to retrieve the account details.
    iv. Move the account record field being processed (account class, first name, or last name), and its length, to Working Storage.
    v. Call `STRSET` to create an unbounded string for the relevant field.

19. The `DO-ACCOUNT-SET-FIRSTNAME` and `DO-ACCOUNT-SET-LASTNAME` generic operations:
    i. Move the length of the first name or last name unbounded strings passed in from the client to Working Storage.
    ii. Call `STRGET` to copy the characters in the unbounded string pointer in Working Storage to the bound string data item in Working Storage.
    iii. Perform the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    iv. Convert the account number in Working Storage to a numeric string.
    v. Move the account number in Working Storage to file.
    vi. Move account first name or last name in Working Storage to file.
    vii. Perform the `UPDATE-ACCOUNT-DETAILS` paragraph, which updates the relevant account record.
20. The `DO-ACCOUNT-GET-ACCOUNTBALANCE` generic operation:
    i. Performs the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    ii. Convert the account number in Working Storage to a numeric string.
    iii. Perform the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file, based on the account number, to retrieve the account details.
21. The `DO-ACCOUNT-MAKE-LODGEMENT` and `DO-ACCOUNT-TRANSFERFUNDSIN` operations:
    i. Perform the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    ii. Convert the account number in Working Storage to a numeric string.

   iii. Perform the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file, based on the account number, to retrieve the account details.
   iv. Calculate the new account balance as the existing account balance plus the temporary amount in Working Storage.
   v. Perform the `UPDATE-ACCOUNT-DETAILS` paragraph, which updates the relevant account record.
   vi. Update the transaction history record for the account, by initializing the transaction history record and calling the `CREATE-TXN-HIST` paragraph, which in turn calls the `GET-TXN-ID` paragraph (to read the transaction number key), the `UPDATE-TXNHIST` paragraph (to update the transaction history record for the account), and the `UPDATE-TXNNUM` paragraph (to update the last-transaction-history-key-used record).

22. The `DO-ACCOUNT-UPDATEADDRESS` operation:
    i. Moves the length of address line 1 to Working Storage.
    ii. Calls `STRGET` to copy the characters in the unbounded string pointer in Working Storage to the string item in Working Storage.
    iii. Moves address line 1 from Working Storage to file.
    iv. Repeats steps i–iii for address line 2 and address line 3.
    v. Performs the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    vi. Converts the account number in Working Storage to a numeric string.
    vii. Moves the account number in Working Storage to file.
    viii. Performs the `UPDATE-ACCOUNT-DETAILS` paragraph, which updates the relevant account record.
23. The `DO-CA-GET-OVERDRAFTLIMIT` operation:
    i. Performs the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    ii. Converts the account number in Working Storage to a numeric string.

iii. Performs the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file, based on the account number, to retrieve the account details.

iv. Moves the overdraft limit from the relevant account record to the operation argument list, and returns this value to the client.

24. The `DO-CA-APPROVENEWOVERDRAFT` operation:

    i. Performs the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.

    ii. Converts the account number in Working Storage to a numeric string.

    iii. Performs the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file, based on the account number, to retrieve the account details.

    iv. Moves the new overdraft amount passed in to the overdraft field on the customer record.

    v. Performs the `UPDATE-ACCOUNT-DETAILS` paragraph, which updates the relevant account record with the new overdraft amount.

25. The `DO-CA-GET-ACCOUNTNUMBER` and `DO-CCA-GET-ACCOUNTNUMBER` operations:

    i. Perform `DO-ACCOUNT-GET-ACCOUNTNUMBER` (see point **16**).

    ii. Assign the account number to the account number parameter of the customer account, and return this to the client.

26. The `DO-CA-GET-ADDR` and `DO-CCA-GET-ADDR` operations:

    i. Perform `DO-ACCOUNT-GET-ADDR` (see point **17**).

    ii. Assign the customer address to the three address parameters of the customer account, and return these to the client. These three string data items are unbounded strings.

27. The `DO-CA-GET-ACCOUNTTYPE` and `DO-CCA-GET-ACCOUNTTYPE` operations:

    i. Perform `DO-ACCOUNT-GET-ACCOUNTTYPE` (see point **18**).

    ii. Assign the relevant account type to the account type parameter of the customer account, and return this to the client. The account type is an unbounded string data item.

28. The `DO-CA-GET-FIRSTNAME` and `DO-CCA-GET-FIRSTNAME` operations:
    i. Perform `DO-ACCOUNT-GET-FIRSTNAME` (see point **18**).
    ii. Assign the relevant first name to the first name parameter of the customer account, and return this to the client. The first name is an unbounded string data item.
29. The `DO-CA-SET-FIRSTNAME` and `DO-CCA-SET-FIRSTNAME` operations:
    i. Assign the unbounded string data item in Working Storage to the first name parameter of the customer account.
    ii. Perform `DO-ACCOUNT-SET-FIRSTNAME` (see point **19**).
30. The `DO-CA-GET-LASTNAME` and `DO-CCA-GET-LASTNAME` operations:
    i. Perform `DO-ACCOUNT-GET-LASTNAME` (see point **18**).
    ii. Assign the relevant last name to the last name parameter of the customer account, and return this to the client. The last name is an unbounded string data item.
31. The `DO-CA-SET-LASTNAME` and `DO-CCA-SET-LASTNAME` operations:
    i. Assign the bounded string data item in Working Storage to the last name parameter of the customer account.
    ii. Perform `DO-ACCOUNT-SET-LASTNAME` (see point **19**).
32. The `DO-CA-GET-ACCOUNTBALANCE` and `DO-CCA-GET-ACCOUNTBALANCE` operations:
    i. Perform `DO-ACCOUNT-GET-ACCOUNTBALANCE` (see point **20**).
    ii. Assign the account balance to the account balance parameter of the customer account, and return this to the client.
33. The `DO-CA-GET-RECENTTRANSACTIONS` and `DO-CCA-GET-RECENTTRANSACTIONS` operations:
    i. Perform `DO-ACCOUNT-GET-RECENTTRAN-D044` (see point **16**).
    ii. Read the transaction history file, using the transaction history key as the key, to check if the specific account has any history records.
    iii. If no transaction history exists, an error message is displayed. Otherwise, `SEQALLOC` is called to create an unbounded sequence and populate it with elements containing details of each history account record, and this sequence is returned to the client.

34. The `DO-CA-MAKELODGEMENT` and `DO-CCA-MAKELODGEMENT` operations:
    i. Move the amount to be lodged to Working Storage.
    ii. Perform `DO-ACCOUNT-MAKELODGEMENT` (see step **21**).
35. The `DO-CA-WITHDRAWFUNDS` and `DO-CCA-WITHDRAWFUNDS` operations:
    i. Move the amount to be withdrawn to Working Storage.
    ii. Perform the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    iii. Convert the account number in Working Storage to a numeric string.
    iv. Perform the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file based on the account key, to retrieve account details.
    v. Calculate the withdrawal limit in Working Storage as the account balance plus the overdraft or credit limit.
    vi. If the amount to be withdrawn is not greater than the withdrawal limit, the server calculates the account balance as the existing account balance minus the amount to be withdrawn, performs the `UPDATE-ACCOUNT-DETAILS` paragraph to update the relevant account record, and performs the `CREATE-TXN-HIST` paragraph to to update the transaction history for the account.
    vii. Conversely, if the amount to be withdrawn is greater than the withdrawal limit, the server calls `COAERR` to raise the `INSUFFICIENT_FUNDS` user exception.
36. The `DO-CA-UPDATEADDRESS` and `DO-CCA-UPDATEADDRESS` operations:
    i. Assign the unbounded string data items in Working Storage to the three address parameters of the customer account.
    ii. Perform `DO-ACCOUNT-UPDATEADDRESS` (see step **22**).
37. The `DO-CA-TRANSFERFUNDSIN` and `DO-CCA-TRANSFERFUNDSIN` operations:
    i. Move amount, to be transferred, to Working Storage.
    ii. Perform `DO-ACCOUNT-TRANSFERFUNDSIN` (see point **21**).

38. The `DO-CA-TRANSFERFUNDSOUT` and `DO-CCA-TRANSFERFUNDSOUT` operations:
    - i. Move amount, to be transferred, to Working Storage.
    - ii. Perform the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    - iii. Convert the account number in Working Storage to a numeric string.
    - iv. Perform the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file based on the account key, to retrieve account details.
    - v. Calculate the withdrawal limit in Working Storage as the account balance plus the overdraft limit.
    - vi. If the amount to be withdrawn is less than the withdrawal limit, the server calculates the account balance as the existing account balance minus the amount to be withdrawn, performs the `UPDATE-ACCOUNT-DETAILS` paragraph to update the relevant account record, and performs the `CREATE-TXN-HIST` paragraph to to update the transaction history for the account.
    - vii. Conversely, if the amount to be withdrawn is greater than the withdrawal limit, the server calls `COAERR` to raise the `INSUFFICIENT_FUNDS` user exception.
39. The `DO-CCA-GET-LIMIT` and `DO-CCA-GET-INTEREST-RATE` operations:
    - i. Perform the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    - ii. Convert the account number in Working Storage to a numeric string.
    - iii. Perform the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file based on the account key, to retrieve account details.
    - iv. Assign the limit or interest rate amount of the customer account to the limit or interest rate parameter, and return this to the client.

40. The `DO-CCA-SET-LIMIT` and `DO-CCA-SET-INTEREST-RATE` operations:
    i. Perform the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    ii. Convert the account number in Working Storage to a numeric string.
    iii. Perform the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file based on the account key, to retrieve account details.
    iv. Move the limit or interest rate passed in to the limit or interest rate field on the customer record.
    v. Perform the `UPDATE-ACCOUNT-DETAILS` paragraph, which updates the relevant account record.
41. The `DO-CCA-CALCULATEINTEREST` operation:
    i. Performs the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    ii. Converts the account number in Working Storage to a numeric string.
    iii. Performs the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file based on the account key, to retrieve account details.
    iv. Checks if the account balance is negative. If so, it calculates the interest due as the account balance multiplied by the interest rate, and places it in the result of the operation’s argument list. Otherwise, it moves zero to the result of the opertion’s argument list.
42. The `DO-CCA-AUTHORISEAMOUNT` operation:
    i. Performs the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
    ii. Converts the account number in Working Storage to a numeric string.

   iii. Performs the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file based on the account key, to retrieve account details.
   iv. Calculates the real limit in Working Storage as the account balance plus the account limit.
   v. Checks if the amount being requested for authorization is within the calculated credit limit. If so, the server calculates an authorization code, by using the COBOL random function, to retrieve a value between 0.0 and 1.0. The result calculated is then multiplied by 10000, to obtain the first four significant digits. The calculated authorization code is assigned to the authorization code parameter of the operation, and then returned to the client. The server performs the `UPDATE-ACOUNT-DETAILS` paragraph to then update the account record.
   vi. Conversely, if the amount being requested for authorization is not within the calculated credit limit, the server calls `COAERR` to raise the `FAILED_TO_AUTHORIZE` user exception.

43. The `DO-CCA-MAKEPURCHASE` operation:
   i. Performs the `GET-OBJECTID-FROM-TARGET` paragraph, which calls `OBJGETID` to retrieve the object name from the related object reference.
   ii. Converts the account number in Working Storage to a numeric string.
   iii. Performs the `RETRIEVE-ACCOUNT-DETAILS` paragraph, which reads the accounts file based on the account key, to retrieve account details.
   iv. Checks to see if the purchase is already authorized (that is, if the authorisation code passed in equals the authorization code on record). If so, the server calculates the account balance as the existing account balance minus the purchase amount, performs the `UPDATE-ACCOUNT-DETAILS` paragraph to update the account record, performs the `CREATE-TXN-HIST` paragraph to update the transaction history for the account, and calls `STRGET` to output the transaction vendor details.

44. A COBOL function that is called to check to see if a system exception has occurred, and to report that system exception, is copied in from the `CHKERRS` copybook.

Writing the Server Mainline

The server mainline module

The next step is to write the server mainline module in which to run the server implementation.

Note: Ordinarily, when you specify the `-S` argument with the Orbix IDL compiler, it generates a module called *`idlmembername`*`SV`, which contains the server mainline code. For the purposes of this demonstration, however, the `FNBSV` module is already provided for you. The `-S` argument is therefore disabled by default in the `FNBIDL` JCL that you use to run the IDL compiler for this demonstration.

Example of the batch FNBSV module

Example 4 shows parts of the batch `FNBSV` module (ellipses are used to denote code omitted for the sake of brevity):

Note: You can find the complete `FNBSV` server implementation program in *`orbixhlq`*`.DEMOS.COBOL.SRC(FNBSV)`.

Example 4: *FNBSV Server Mainline Module (Sheet 1 of 11)*

```
****************************************************************
*  Copyright (c) 2001-2003 IONA Technologies PLC.
*  All Rights Reserved.
*
* Description: This is the batch server mainline of the FNB
*              demo.
*
****************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID.           FNBSV.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    …

        COPY IORSLCT REPLACING
            "X-IOR" BY ACCMGR-IOR
            "X-IORFILE" BY "IORFILE"
            "X-IOR-STAT" BY ACCMGR-IOR-STAT.

DATA DIVISION.
```

1 (INPUT-OUTPUT SECTION.)
2 (COPY IORSLCT REPLACING)

Example 4: *FNBSV Server Mainline Module (Sheet 2 of 11)*

```
    FILE SECTION.

3   COPY FNBRECS.
4   COPY IORFD REPLACING
                 "X-IOR" BY ACCMGR-IOR
                 "X-REC" BY ACCMGR-REC.


    WORKING-STORAGE SECTION.

5   COPY NAMES.
6   COPY FNB.
7   COPY CORBA.

    …

8   COPY PROCPARM.

9       PERFORM OPEN-FILE.

10      CALL "ORBSTAT" USING ORBIX-STATUS-INFORMATION.
        SET WS-ORBSTAT TO TRUE.
        PERFORM CHECK-STATUS.

11      CALL "ORBARGS" USING ARG-LIST
                             ARG-LIST-LEN
                             ORB-NAME
                             ORB-NAME-LEN.
        SET WS-ORBARGS TO TRUE.
        PERFORM CHECK-STATUS.

12      CALL "ORBSRVR" USING SERVER-NAME
                             SERVER-NAME-LEN.
        SET WS-ORBSRVR TO TRUE.
        PERFORM CHECK-STATUS.


    ****************************************************************
    * Interface Section Block
    ****************************************************************

    * Generating IOR for interface bankobjects/AccountMgr
13      CALL "ORBREG" USING ACCMGR-INTERFACE.
        SET WS-ORBREG TO TRUE.
        PERFORM CHECK-STATUS.
```

Example 4: *FNBSV Server Mainline Module (Sheet 3 of 11)*

```
         OPEN OUTPUT ACCMGR-IOR.
         IF ACCMGR-IOR-STAT NOT = 0
             GO TO EXIT-PRG
         END-IF.

14       CALL "OBJNEW" USING SERVER-NAME
                           INTERFACE-NAME OF INTERFACE-NAMES-ARRAY(1)
                            OBJECT-IDENTIFIER OF OBJECT-ID-ARRAY(1)
                            ACCMGR-OBJ.
         SET WS-OBJNEW TO TRUE.
         PERFORM CHECK-STATUS.

15       CALL "OBJTOSTR" USING ACCMGR-OBJ
                              IOR-REC-PTR.
         SET WS-OBJTOSTR TO TRUE.
         PERFORM CHECK-STATUS.

16       CALL "STRGET" USING IOR-REC-PTR
                            IOR-REC-LEN
                            ACCMGR-REC.
         SET WS-STRGET TO TRUE.
         PERFORM CHECK-STATUS.

17       CALL "STRFREE" USING IOR-REC-PTR.
         SET WS-STRFREE TO TRUE.
         PERFORM CHECK-STATUS.

18       WRITE ACCMGR-REC.
         IF ACCMGR-IOR-STAT NOT = 0 THEN
             GO TO EXIT-PRG
         END-IF.

         CLOSE ACCMGR-IOR.
         IF ACCMGR-IOR-STAT NOT = 0 THEN
             GO TO EXIT-PRG
         END-IF.

     * Register interface bankobjects/Account
19       CALL "ORBREG" USING ACCOUNT-INTERFACE.
         SET WS-ORBREG TO TRUE.
         PERFORM CHECK-STATUS.

     * Register interface bankobjects/CurrentAccount
19       CALL "ORBREG" USING CA-INTERFACE.
```

Example 4: *FNBSV Server Mainline Module (Sheet 4 of 11)*

```
         SET WS-ORBREG TO TRUE.
         PERFORM CHECK-STATUS.

     * Register interface bankobjects/CreditCardAccount
19       CALL "ORBREG" USING CCA-INTERFACE.
         SET WS-ORBREG TO TRUE.
         PERFORM CHECK-STATUS.

     * Register interface NamingContextExt for access as a
     * CORBA Interace
20       CALL  "ORBREG"   USING COSNAMING-NAMIN-EF2D-INTERFACE.
         SET WS-ORBREG TO TRUE.
         PERFORM CHECK-STATUS.

     * Attain a reference to the Naming service using OBJRIR
         DISPLAY "Attaining reference to the Naming Service".
         SET NAMING-SERVICE TO TRUE.
21       CALL  "OBJRIR"   USING SERVICE-REQUESTED
                                NAME-SERVICE-OBJ.
         SET WS-OBJRIR TO TRUE.
         PERFORM CHECK-STATUS.


     * Bind acc mgr object reference to the naming service
     * setting up values
     * set ID to object to resolve
         MOVE SPACES TO WS-THE-STRING.
         MOVE "BankObjects_AccountMgr" TO WS-THE-STRING.
         MOVE LENGTH OF WS-THE-STRING TO
             WS-THE-STRING-LENGTH.

22       CALL "STRSET"    USING IDL-ID OF N OF N-1 OF
                                COSNAMING-NAMINGCONT-330B-ARGS
                                WS-THE-STRING-LENGTH
                                WS-THE-STRING.
         SET WS-STRSET TO TRUE.
         PERFORM CHECK-STATUS.
     * set kind to nothing
         MOVE SPACES TO WS-THE-STRING.
         MOVE 1 TO WS-THE-STRING-LENGTH.
22       CALL "STRSET"    USING KIND OF N OF N-1 OF
                                COSNAMING-NAMINGCONT-330B-ARGS
                                WS-THE-STRING-LENGTH
                                WS-THE-STRING.
         SET WS-STRSET TO TRUE.
```

Example 4: *FNBSV Server Mainline Module (Sheet 5 of 11)*

```
        PERFORM CHECK-STATUS.


    * A sequence of name components is used by cosnaming to describe
    * a path in the naming service graph.
    * For demo purposes, just branch off root

23      MOVE 1 TO WS-SEQUENCE-LENGTH.
        SET COSNAMING-NAME TO TRUE.
24      CALL "SEQALLOC"   USING WS-SEQUENCE-LENGTH
                                NAMES-TYPE
                                NAMES-TYPE-LENGTH
                                N-SEQUENCE OF
                                COSNAMING-NAMINGCONT-330B-ARGS.
        SET WS-SEQALLOC TO TRUE.
        PERFORM CHECK-STATUS.

        MOVE 1  TO  SEQUENCE-MAXIMUM   OF N-SEQUENCE OF
                                COSNAMING-NAMINGCONT-330B-ARGS.
        MOVE 1  TO SEQUENCE-LENGTH     OF N-SEQUENCE OF
                                COSNAMING-NAMINGCONT-330B-ARGS.

25      CALL "SEQSET"     USING N-SEQUENCE OF
                                COSNAMING-NAMINGCONT-330B-ARGS
                                WS-SEQUENCE-LENGTH
                                N-1 OF
                                COSNAMING-NAMINGCONT-330B-ARGS.
        SET WS-SEQSET TO TRUE.
        PERFORM CHECK-STATUS.
    * set obj value
26      SET OBJ OF COSNAMING-NAMINGCONT-330B-ARGS TO
                   ACCMGR-OBJ.
    * Bind acc mgr object reference to the naming service
        DISPLAY "Trying to Bind to the Naming Service...".

        SET COSNAMING-NAMINGCONTEXTEX-330B TO TRUE

27      CALL "ORBEXEC"   USING NAME-SERVICE-OBJ
                               COSNAMING-NAMIN-EF2D-OPERATION
                               COSNAMING-NAMINGCONT-330B-ARGS
                               NAMES-USER-EXCEPTIONS.
        SET WS-ORBEXEC TO TRUE.
        PERFORM CHECK-STATUS.

28  * already bound exception, call rebind
```

Example 4: *FNBSV Server Mainline Module (Sheet 6 of 11)*

```
     IF D OF NAMES-USER-EXCEPTIONS = 4
         SET D-NO-USEREXCEPTION OF NAMES-USER-EXCEPTIONS TO TRUE
         DISPLAY "Already bound exception Thrown… "
         DISPLAY "Trying to Rebind to the Naming Service… "

* Rebind acc mgr object reference to the naming service
* setting up values
* set ID to object to resolve
         MOVE SPACES TO WS-THE-STRING
         MOVE "BankObjects_AccountMgr" TO WS-THE-STRING
         MOVE LENGTH OF WS-THE-STRING TO
             WS-THE-STRING-LENGTH

         CALL "STRSET"    USING IDL-ID OF N OF N-1 OF
                          COSNAMING-NAMINGCONT-A492-ARGS
                          WS-THE-STRING-LENGTH
                          WS-THE-STRING
         SET WS-STRSET TO TRUE
         PERFORM CHECK-STATUS
* set kind to nothing
         MOVE SPACES TO WS-THE-STRING
         MOVE 1 TO WS-THE-STRING-LENGTH
         CALL "STRSET"    USING KIND OF N OF N-1 OF
                          COSNAMING-NAMINGCONT-A492-ARGS
                          WS-THE-STRING-LENGTH
                          WS-THE-STRING
         SET WS-STRSET TO TRUE
         PERFORM CHECK-STATUS

* A sequence of name components is used by cosnaming to describe
* a path in the naming service graph.
* For demo purposes, just branch off root

         MOVE 1 TO WS-SEQUENCE-LENGTH
         SET COSNAMING-NAME TO TRUE
         CALL "SEQALLOC"    USING WS-SEQUENCE-LENGTH
                                  NAMES-TYPE
                                  NAMES-TYPE-LENGTH
                                  N-SEQUENCE OF
                                  COSNAMING-NAMINGCONT-A492-ARGS
         SET WS-SEQALLOC TO TRUE
         PERFORM CHECK-STATUS

         MOVE 1  TO  SEQUENCE-MAXIMUM  OF N-SEQUENCE OF
                              COSNAMING-NAMINGCONT-A492-ARGS
```

Example 4: *FNBSV Server Mainline Module (Sheet 7 of 11)*

```
        MOVE 1  TO   SEQUENCE-LENGTH  OF N-SEQUENCE OF
                              COSNAMING-NAMINGCONT-A492-ARGS

        CALL "SEQSET"    USING N-SEQUENCE OF
                               COSNAMING-NAMINGCONT-A492-ARGS
                               WS-SEQUENCE-LENGTH
                               N-1 OF
                               COSNAMING-NAMINGCONT-A492-ARGS
        SET WS-SEQSET TO TRUE
        PERFORM CHECK-STATUS
* set obj value
        SET OBJ OF COSNAMING-NAMINGCONT-A492-ARGS TO
                ACCMGR-OBJ
* Rebind acc mgr object reference to the naming service

        SET COSNAMING-NAMINGCONTEXTEX-A492 TO TRUE

        CALL "ORBEXEC"   USING NAME-SERVICE-OBJ
                               COSNAMING-NAMIN-EF2D-OPERATION
                               COSNAMING-NAMINGCONT-A492-ARGS
                               NAMES-USER-EXCEPTIONS
        SET WS-ORBEXEC TO TRUE
        PERFORM CHECK-STATUS

        IF NOT D-NO-USEREXCEPTION OF NAMES-USER-EXCEPTIONS
            PERFORM BIND-EVAL
        ELSE
            DISPLAY "Rebind Success…"
        END-IF

* clean up after ourselves
29      CALL "SEQFREE"    USING N-SEQUENCE OF
                                COSNAMING-NAMINGCONT-A492-ARGS
        SET WS-SEQFREE TO TRUE
        PERFORM CHECK-STATUS
    ELSE
        IF NOT D-NO-USEREXCEPTION OF NAMES-USER-EXCEPTIONS
            PERFORM BIND-EVAL
        ELSE
            DISPLAY "Bind Success…"
        END-IF
* clean up after ourselves
        CALL "SEQFREE"    USING N-SEQUENCE OF
                                COSNAMING-NAMINGCONT-300B-ARGS
        SET WS-SEQFREE TO TRUE
```

Example 4: *FNBSV Server Mainline Module (Sheet 8 of 11)*

```
        PERFORM CHECK-STATUS
    END-IF.

    DISPLAY "Giving control to the ORB to process Requests".

30  CALL "COARUN".
    SET WS-COARUN TO TRUE.
    PERFORM CHECK-STATUS.

 EXIT-PRG.
31  PERFORM CLOSE-FILE.

    DISPLAY " Bank shutting down… "
    CALL "SHUTDOWN".
    STOP RUN.

****************************************************************
* Check Errors Copybook
****************************************************************
 COPY CHKERRS.

 OPEN-FILE.
*=============

    OPEN I-O ACCOUNTS.
    IF ACCOUNT-STATUS NOT = 0
        IF ACCOUNT-STATUS = 97
            DISPLAY '*** integrity check successful '
                    'server starting ...'
            DISPLAY '*** previous shutdown without file '
                    ' closure detected. '
            DISPLAY '*** In future use MVS stop (/p) to stop '
                    'server in orderly way'
        ELSE
            DISPLAY '*** A response of, ' ACCOUNT-STATUS ', was '
                    'detected when opening the Account file.'
            GOBACK
        END-IF
    END-IF.

    OPEN I-O TXNHIST.
    IF TXNHIST-STATUS NOT = 0
        IF TXNHIST-STATUS = 97
            DISPLAY '*** integrity check successful '
                    'server starting ...'
```

Example 4: *FNBSV Server Mainline Module (Sheet 9 of 11)*

```
             DISPLAY '*** previous shutdown without file '
                     ' closure detected. '
             DISPLAY '*** In future use MVS stop (/p) to stop '
                     'server in orderly way'
         ELSE
            DISPLAY '*** A response of, ' TXNHIST-STATUS ', was '
                     'detected when opening the TXNHIST file.'
             GOBACK
         END-IF
     END-IF.
     OPEN I-O TXNNUM.
     IF TXNNUM-STATUS NOT = 0
         IF TXNNUM-STATUS = 97
             DISPLAY '*** integrity check successful '
                     'server starting ...'
             DISPLAY '*** previous shutdown without file '
                     ' closure detected. '
             DISPLAY '*** In future use MVS stop (/p) to stop '
                     'server in orderly way.'
         ELSE
             DISPLAY '*** A response of, ' TXNNUM-STATUS ', was '
                     'detected when opening the Txnnum file.'

             GOBACK
         END-IF
     END-IF.

     EXIT.

  CLOSE-FILE.
 *=============

     CLOSE ACCOUNTS.
     IF ACCOUNT-STATUS NOT = 0
         DISPLAY '*** A response of, ' ACCOUNT-STATUS ', was '
                 'detected when closing the Account file.'
     END-IF.
     CLOSE TXNHIST.
     IF TXNHIST-STATUS NOT = 0
         DISPLAY '*** A response of, ' TXNHIST-STATUS ', was '
                 'detected when closing the TXNHIST file.'
     END-IF.

     CLOSE TXNNUM.
     IF TXNNUM-STATUS NOT = 0
```

Example 4: *FNBSV Server Mainline Module (Sheet 10 of 11)*

```
        DISPLAY '*** A response of, ' TXNNUM-STATUS ', was '
                'detected when closing the Txnnum file.'
    END-IF.

    EXIT.

*****************************************************************
*
* Bind Evaluate checks if a user exception is returned from
* the bind/rebind operation and deals with the user exception if
* one is thrown.
*
*****************************************************************
 BIND-EVAL.
*===========
    EVALUATE TRUE

* CannotProceed exception thrown
    WHEN D-COSNAMING-NAMINGCONTEXT-9F29 OF
            NAMES-USER-EXCEPTIONS
        DISPLAY "Bind Unsuccessful ……"
        MOVE SPACES TO WS-EXCEPTION-STRING
        MOVE EX-COSNAMING-NAMINGCONTEX-1482 TO
              WS-EXCEPTION-STRING-LEN
        PERFORM THROW-USER-EXCEPTION

* InvalidName exception thrown
    WHEN D-COSNAMING-NAMINGCONTEXT-29EC OF
           NAMES-USER-EXCEPTIONS
        DISPLAY "Bind Unsuccessful ……"
        MOVE SPACES TO WS-EXCEPTION-STRING
        MOVE EX-COSNAMING-NAMINGCONTEX-9079 TO
            WS-EXCEPTION-STRING-LEN
        PERFORM THROW-USER-EXCEPTION

    END-EVALUATE.

*****************************************************************
*
* Print exception message
*
*****************************************************************

 THROW-USER-EXCEPTION.
*====================
```

Example 4: *FNBSV Server Mainline Module (Sheet 11 of 11)*

```
    CALL "STRGET"  USING  EXCEPTION-ID OF
                          NAMES-USER-EXCEPTIONS
                          WS-EXCEPTION-STRING-LEN
                          WS-EXCEPTION-STRING.
    SET WS-STRGET TO TRUE.
    PERFORM CHECK-STATUS.

    DISPLAY "Exception ID : " WS-EXCEPTION-STRING.

    CALL "STRFREE"  USING  EXCEPTION-ID OF
                           NAMES-USER-EXCEPTIONS.
    SET WS-STRFREE TO TRUE.
    PERFORM CHECK-STATUS.

    MOVE 12 TO RETURN-CODE.
    GO TO EXIT-PRG.
```

Explanation of the batch FNBSV module

The FNBSV module can be explained as follows:

1. This section defines the files to be used by the server application for storing account data, transaction history data, and last-transaction-history-key-used-per-account data.
2. The COBOL SELECT statement entry for file processing, for use with the COPY…REPLACING statement, is copied from the IORSLCT copybook.
3. The record layouts for storing account data, transaction history data, and last-transaction-history-key-used-per-account data are copied from the FNBRECS copybook.
4. The COBOL FD statement entry for file processing, for use with the COPY…REPLACING statement, is copied from the IORFD copybook.
5. Data definitions used for working with operation parameters and return values for each Naming Service interface, defined in the COSNAMI IDL member, are copied from the NAMES copybook.
6. Data definitions used for working with operation parameters and return values for each FNB server interface, defined in the FNB IDL member, are copied from the FNB copybook.

7. Various Orbix COBOL definitions, such as `REQUEST-INFO` used by the `COAREQ` function, and `ORBIX-STATUS-INFORMATION` which is used to register and report system exceptions raised by the COBOL runtime, are copied from the `CORBA` copybook.
8. The appropriate definitions to allow the program to accept parameters for use with the `ORBARGS` call are copied from the `PROCPARM` copybook.
9. The `OPEN-FILE` paragraph is performed to open the `ACCOUNTS`, `TXNHIST`, and `TXNNUM` data sets.
10. `ORBSTAT` is called to register the `ORBIX-STATUS-INFORMATION` block that is contained in the `CORBA` copybook. Registering the `ORBIX-STATUS-INFORMATION` block allows the COBOL runtime to populate it with exception information, if necessary.
11. `ORBARGS` is called to initialize a connection to the ORB, and to read the command-line arguments to the program, which are specified as parameters on the PPARM JCL parameter.
12. `ORBSRVR` is called to set the server name.
13. `ORBREG` is called to register the `AccountMgr` interface with the Orbix COBOL runtime.
14. `OBJNEW` is called to create a persistent server object of the `AccountMgr` type. The object reference created encapsulates the specified object ID and interface name.
15. `OBJTOSTR` is called to translate the object reference created by `OBJNEW` into a stringified IOR. The stringified IOR is then written to the `IORFILE` member.
16. `STRGET` is called to copy the characters in the unbounded stringified IOR to a bounded string.
17. `STRFREE` is called to release the dynamically allocated memory for the unbounded stringified IOR.
18. The account manager IOR is written to file.
19. `ORBREG` is called to register the `Account`, `CurrentAccount`, and `CreditCardAccount` interfaces respectively with the Orbix COBOL runtime.

20. `ORBREG` is called again to register the `NamingContextExt` interface with the Orbix COBOL runtime, so that it can be accessed as a CORBA interface.
21. `OBJRIR` is called to obtain an object reference to the Naming Service.
22. `STRSET` is called to set the `id` and `kind` fields of the sequence member for the name sequence that is now about to be built.
23. A sequence of length `1` is allocated.
24. `SEQALLOC` is called to allocate initial storage for the sequence.
25. `SEQSET` is called to create the first sequence element.
26. Set the account manager object that you want to bind into the Naming Service.
27. `ORBEXEC` is called to allow for invocations on the server interface represented by the supplied object reference.
28. If the already bound exception is thrown, a rebind is attempted and steps 22–27 are then repeated.
29. `SEQFREE` is called to release the name sequence.
30. `COARUN` is called, to enter the `ORB::run` loop, to allow the ORB to receive and process client requests.
31. The `CLOSE-FILE` paragraph is called to close the `ACCOUNT`, `TXNHIST`, and `TXNNUM` data sets.

Building the Server

Overview

This section describes how to build the FNB COBOL server.

Note: The server is not supplied pre-built, so you must complete the steps described in this section.

Before building the server

Before you build the server ensure that you have completed the steps described in "Generating COBOL copybooks for Naming Service" on page 32 and "Generating COBOL copybooks for the FNB server" on page 32.

JCL to build the server

Sample JCL used to compile and link the FNB back-end server mainline and server implementation is in `*orbixhlq*.DEMOS.COBOL.BLD.JCL(FNBSB)`. When this JCL has successfully executed, it results in a load module that is contained in `*orbixhlq*.DEMOS.COBOL.LOAD(FNBSV)`.

Running the FNB COBOL Back-End Server

This chapter describes how to start the COBOL back-end server component of the FNB demonstration.

Note: You must start the back-end server on OS/390 before you start the front-end and middle-tier components on Windows or UNIX. After you have completed this chapter see the *First Northern Bank Tutorial* supplied with Orbix for details of how to start the front end and middle tier.

In this chapter

This chapter discusses the following topics:

Prerequisites

Overview

This section describes what you need to do before you can actually start the FNB COBOL back-end server on OS/390.

Note: See the *Mainframe Installation Guide* for more details about customizing various services such as the Naming Service.

In this section

This section discusses the following topics:

Creating the VSAM data sets

Overview

As explained in "Back-tier CORBA server" on page 5, the FNB COBOL server uses four VSAM data sets for object data persistence. Before you can start the server and run the FNB demonstration these data sets must be created.

Summary of data sets

To recap, the four data sets used store the following data:

- Account data—this includes an alternate index, to allow for referencing account records by account number or account type.
- Transaction history.
- Last used account number.
- Last used transaction history key (for each account).

JCL to create the data sets

You can use the JCL in `*orbixhlq*.DEMOS.COBOL.RUN.JCL(FNBVSAM)` to create these VSAM data sets.

Note: An IEC161I rc 39 with VSAM error code 100 is generated when you submit the `FNBVSAM` JCL. This error is normal and can be ignored.

Starting the Orbix Locator Daemon

Overview

An Orbix locator daemon must be running on the server's location domain before you try to run the server application. The Orbix locator daemon is a program that implements several components of the ORB, including the Implementation Repository. The locator runs in its own address space on the server host, and provides services to the client and server, both of which need to communicate with it.

When you start the Orbix locator daemon, it appears as an active job waiting for requests. See the *CORBA Administrator's Guide* for more details about the locator daemon.

JCL to start the Orbix locator daemon

If the Orbix locator daemon is not already running, you can use the JCL in `*orbixhlq*.JCL(LOCATOR)` to start it.

Locator daemon configuration

The Orbix locator daemon uses the Orbix configuration member for its settings. The JCL that you use to start the locator daemon uses a sample configuration member that is provided in `*orbixhlq*.CONFIG(DEFAULT@)`.

Starting the Orbix Node Daemon

Overview

An Orbix node daemon must be running on the server's location domain before you try to run the server application. The node daemon acts as the control point for a single machine in the system. Every machine that will run an application server must be running a node daemon. The node daemon monitors and manages the application servers running on that machine. The locator daemon relies on the node daemons to start processes and inform it when new processes have become available.

When you start the Orbix node daemon, it appears as an active job waiting for requests. See the *CORBA Administrator's Guide* for more details about the node daemon.

JCL to start the Orbix node daemon

If the Orbix node daemon is not already running, you can use the JCL in `orbixhlq.JCL(NODEDAEM)` to start it.

Node daemon configuration

The Orbix node daemon uses the Orbix configuration member for its settings. The JCL that you use to start the node daemon uses a configuration member that is provided in `orbixhlq.CONFIG(DEFAULT@)`.

Starting the Naming Service

Overview

The Naming Service maintains a database of names and the objects associated with them. An association between a name and an object is called a *binding*. The IDL interfaces to the Naming Service provide operations to access the database of bindings. For example, you can create new bindings, resolve names, and delete existing bindings.

IONA's implementation of the Naming Service is implemented as a normal Orbix server. This server contains objects that support the standard IDL interfaces to the Naming Service. These interfaces are defined in `*orbixhlq*.INCLUDE.OMG.IDL(COSNAMI)`. See the *CORBA Programmer's Guide, C++* for more details about the Naming Service.

JCL to start the Naming Service

If the Naming Service is not already running, you can use the JCL in `*orbixhlq*.JCL(NAMING)` to start it.

Naming Service configuration

The Naming Service uses the Orbix configuration member for its settings. The JCL that you use to start the Naming Service uses a configuration member that is provided in `*orbixhlq*.CONFIG(DEFAULT@)`.

Starting the Server

Overview

This section describes how to run the FNB COBOL back-end server. The following topics are discussed:

- "JCL to run the server" on page 83.
- "JCL to publish the Naming Service IOR" on page 83.

JCL to run the server

To run the supplied `FNB` server application, submit the following JCL:

```
orbixhlq.DEMOS.COBOL.RUN.JCL(FNBSV)
```

Note: You should use the OS/390 `STOP (/P)` operator command to subsequently stop the server. Otherwise, the server cannot close the VSAM data sets and will issue a warning the next time it tries to open them.

When you run the server, the object reference of the AccountManager factory object is automatically published in the Naming Service.

JCL to publish the Naming Service IOR

To allow the FNB demonstration mid-tier to access the COBOL back-end server, the client must be able to obtain the IOR for the Naming Service. To publish the IOR for the Naming Service, submit the following JCL:

```
orbixhlq.DEMOS.COBOL.RUN.JCL(FNBNSIOR)
```

This writes the IOR for the Naming Service to *`orbixhlq`*`.DEMOS.IOR(NS)`.

After Starting the Server

Overview

This section describes two extra steps that must be completed after you have started the FNB COBOL server on OS/390 but before you start the front-end and middle-tier components of the FNB demonstration on Windows or UNIX. These steps are essential to ensure that the front-end and middle-tier components can successfully contact the mainframe server.

Note: After you have completed these two steps, the front-end and middle-tier components can be started as normal, as described in the *First Northern Bank Tutorial* that is supplied with Orbix.

The following topics are discussed:

- "Copying Naming Service IOR to Windows or UNIX" on page 84.
- "Contacting the Mainframe Naming Service" on page 84.

Copying Naming Service IOR to Windows or UNIX

The IOR member in *orbixhlq*.DEMOS.IOR(NS) is a simple text file that contains the IOR for the back-end Naming Service on OS/390.

You must copy this IOR file to the Windows or UNIX host where you have installed the front-end and middle-tier components of the FNB demonstration. You should copy the IOR file to the *install-dir*/asp/6.0/demos/common/fnb directory, where *install-dir* represents the full path to your installation directory.

Contacting the Mainframe Naming Service

After you have copied the IOR for the back-end Naming Service to the relevant Windows or UNIX host, enter the following command on that host, in the *install-dir*/asp/6.0/demos/common/fnb directory:

```
itant -Dmainframe_ns_ior_file=mainframe.ior
   add_federated_mainframe
```

Note: Even though you can give the IOR file any name, you should call it something meaningful, such as mainframe.ior in the preceding example.

The preceding command allows the middle-tier client to subsequently contact the Naming Service on the OS/390 backend instead of the local Naming Service on its own host.

At this stage, the front-end and middle-tier components of the FNB demonstration can now be started on Windows or UNIX. See the *First Northern Bank Tutorial* that is supplied with Orbix for details of how to start these components.