User’s Guide

A Framework for Object Persistency for GNAT

Version 0.6.1

Document Revision $Revision: 1.16 $

Michael Erdmann

Edited by

Michael Erdmann

User’s Guide: A Framework for Object Persistency for GNAT; Version 0.6.1; Document Revision $Revision: 1.16 $

by Michael Erdmann

Edited by Michael Erdmann

Revision History

Revision $Revision: 1.16 $ $Date: 2003/10/12 14:05:05 $ Revised by: $Author: merdmann $

Table of Contents

List of Tables

List of Examples

Preface

This package is part of the GNADE project hosted at http://gnade.sourceforge.net. The build environment is based upon the ABE project hosted at http://ascl.sourceforge.net.

The objective of this project is to provide a basic approach to object persistency for Ada 95. The project will be done in two phases:

- Phase 1 - Objects are stored in the filesystem
- Phase 2 - Objects are stored in an RDBMS

This document represents Phase 1. In this phase some basic features of an object database as e.g. collections will be available.

The coordination of the development work is done by:

Michael Erdmann
<Michael.Erdmann@snafu.de>

I. Introduction

Chapter 1. Overview

Object persistency means that your application may create a a data object which survives auto magically the shutdown of your application. The next time you start your data object is available again for use. Normally this is achieved by the explicit use of files or data bases, where the data is stored. In such cases effort has to be spend to read/write object from the files system or to locate the objects in the databases and to construct the objects from the data.

Lets assume the following code fragment:

```
declare
   V : Persistent_Type ;
begin
   ..... code invoking the variable V
end ;
```

Persistence in this context mean, that the state of the object V is available again, when next time the block is entered. In order to implement persistency in this context is it sufficient simply to the V a globally defined variable.

The persistency concept provided by ODB works in a very similar way. All persistent objects are derived from a basic data types. Objects of this type are stored in a special storage pool which is saved at application termination and loaded when the application using the objects is starting up. A typical code fragment looks like this:

```
declare
   V : Reference := Person.Create( "Michael Erdmann" ) ;
begin
   ..... code invoking the variable V
end ;
```

The function Person.Create creates an new instance with the name Michael Erdmann if the instance is not yet stored in the in the persistent storage pool. If the object is already existing, the procedure returns the reference to the already existing object. This show one of the key concepts of ODB, in order to make an Object persistent it has to be named, because otherwise it would not be possible to retrieve the object.

In order to make the concept of persistency easy to handle, oDB provides a preprocessor which adds automatically all code to support persistency of a type to an Ada95 package. A typical construct may look like this:

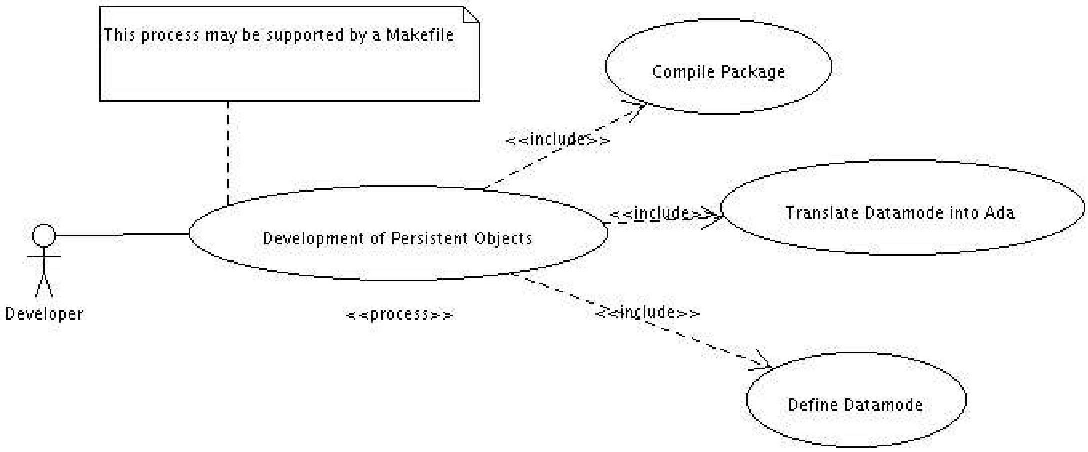
```
     package Person is
        ..............
        type Object is persistent record
               Id   : attribute Natural := 0;
               Name : attribute Unbounded_String;
               Tmp  : Natural;
            end record;

 function Create(
          Name   : in String ) return Reference is
```

```
            Result : Reference := Lookup_Object( Name );
            H      : Handle := Handle( Result );
         begin
            if Result = null then
                Put_Line("New instance of person created" );
                Result := new Object;
                Name_Object( Result, Name );

                H := Handle( Result );

                H.Name := To_Unbounded_String(Name);
                H.Id   := 1;
             end if;
             return Result;
          end Create;

         ...............
      end Person;
```

This construct defines in the package Person a persistent object Object with the attributes Id and Name. Attributes are fields which are stored. All other fields will not be restored when the object is restored.

The procedure Create first checks if an object with the given name is already knwon. If not, a new instance will be created, otherwise the already exisisting instance will be used.

The development cycle which leads to an application using persistency is shown in the diagram below:



Steps involved in building ODB applications

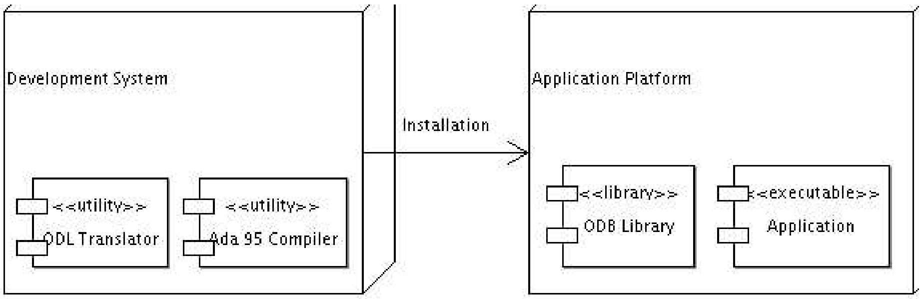
In the first step, the developer implements a persistent class by creating a package which contains a single persistent data type type defining the so called data model using a simple extension of the Ada 95 language. The example below shows such a code fragement:

```
   type Object is persistent record
          A,X,Y   : attribute Integer ;                -- this goes into the DB
          B       : attribute String(1..50) ;          -- this also
          C       : Integer := 2222;                   -- this not
```

```
end record ;
```

This code will be translated by the ODL preprocessor into Ada 95 code which contains the Serialize/Deserialize and Factory code.

After the ODL translation, the code is compiled using an Ada 95 compiler and is ready for use.



Deployment of the ODB components

Chapter 2. Getting started

The ODB environment is distributed as source package, which means in order to use is, the software has to be build (compiled and linked) on your system before it can be used.

Installation on Unix like systems

After you obtained the source code from the net you need to install and compile it. This chapter describes this first steps of installing the environment onto your system.

Unpacking the distribution

The source code is normally distributed as compressed tar file. To unpack the distribution execute the command:

```
gunzip -c tarfilename.gz | tar xvf -
```

This will unpack the directory tree of the development environment.

Configuring the ODB installation

After unpacking the distribution change into the top level directory of the ODB release. Before you run the configure script examine the contents of the file etc/config.site.

After checking the settings, the environment has to be configured by means of the command:

```
cd odb-src-....
./configure [ option(s) ]
```

This command will perform the actual configuration by checking for requiered software components and installation options.

Compiling the distribution

To build the environment enter the command below:

```
gmake all
```

Installing globally on the target system

Currently there is no automatic procedure available. You need to copy the compolents from the following directories manually into some reasonable directories.

```
linux-gnu-i686-bin
linux-gnu-i686-doc
linux-gnu-i686-include
linux-gnu-i686-install
linux-gnu-i686-lib
```

GPS Support

This release contains some files to support the GPS environment from ACT. The source root directory contains a file which is called gnu.xml which contains extensions in the GPS menus allowing you to configure and to compile the release from the GPS environment. This file should be installed locally in the directory .gps/customize or in the installation directory of GPS as described in the documents.

The HTML based documentation is also made available via GPS if the environment variable GPS_DOC_PATH contains the place where ODB is installed.

Chapter 3. ODB Basics

This section gives an brief overview about the ODB package from the programers point of view

An object is an instance of a class. A class is assumed to be implemented as an Ada 95 package which exports a data type and operations on this data type.

Object Model of ODB

Persistent objects are always derived from the type ODB.Persistent.Object. All persistent objects are allocated in dedicated storage pool. Each object in the storage pool is linked to a so called object table which is contained in the ODB.Persistent package.

Since the type Persistent.Object is abstract, the implementation for the following methods has to be provided by any implementation of persistent object:

- Factory
- Serialize
- Deserialize

The function Factory return a pointer to an instance of the class.

Serialize writes out the object attributes into a memory stream and at the same time it has to setup the header information of an instance. The header contains a list of all attributes stored in the object and the corresponding offsets.

Object Life Cycle

In order to make an persistent object really persistent, the object has to be named. Otherwise, the application would not be able to retrieve the object.

Loading objects

Upon startup of an application using ODB, all types which are intended to be stored in the object store are registered in the ODB.Persistent package. Together with the type name, the pointer to a procedure is stored. This procedure is called Factory and is used to create an object of the given type in the memory.

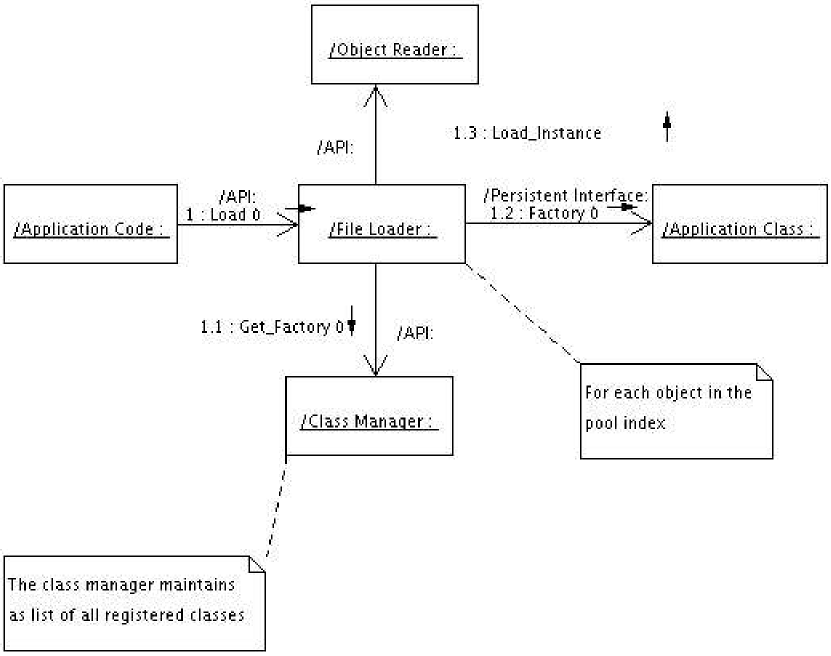
When reading in an object from the object store, the type name is looked up in the list of all registered factories and the factory is called to create the actual instance.

```
function Factory return Persistent.Reference is
   Result : Reference := new Object;
begin
   Handle( Result ).Data := new Object_Data_Type;
   return Result;
end;
```

Important to note is that the new operation is used together with the Reference type. This forces the object to be allocated in the storage_pool of the object management.

In addition to the registration of the Factory for the class, the attributes have to be declared for the object implementation.

Attribute names are used to map fields of an Ada 95 object between data entries in the storage. As a consequence fields may be added to the object during development of the application and the object stay still loadable.



The sequence of calls when loading an object from the object

Object are loaded from the storage by means of the Deserialize proceudure. This is an abstract procedure which has to be provided by the implementation:

```
procedure Deserialize(
   Item   : in out Object;
   Header : in Storage_Header.Handle;
   S      : in Stream_Access );
```

The purpose of this function is to read in the object attributes from the given stream. The storage_header contains the fields and the offset of the attributes within the memory stream.

```
procedure Deserialize(
   Item   : in out Object;
   Header : in Storage_Header.Handle;
   S      : in Stream_IO.Stream_Access ) is
   Field  : String_Array.Handle := Attributes( Header.all );
begin
   for i in Field'Range loop
      declare
```

```
            ID     : Natural;
            Offset : Natural;
            Name   : constant String := To_String( Field(i) );
          begin
             ID := Classes.Attribute( Object'Tag, Name );
             if ID /= 0 then
                Offset := Storage_Header.Lookup_Attribute( Header.all, Name );
                Read_Offset( S, Offset );

                case ID is
                   when D_Name  =>
                      Item.Name := Unbounded_String'Input(S);

                   when D_Street =>
                      Item.Street := Unbounded_String'Input(S);

      ...........

                   when Others =>
                      null;
                end case;
             end if;
          exception
             when Storage_Header.Unknown_Attribute =>
                null;
          end;
       end loop;

       String_Array.Free( Field );
    end Deserialize;
```

This procedure reads in all attributes which have been listed in the object header. For each field in the header registered field id and the offset in the object storage is looked up. The read pointer is set to the found offset and the data type is read in. If a attribute name is not known in the class the field will be ignored.

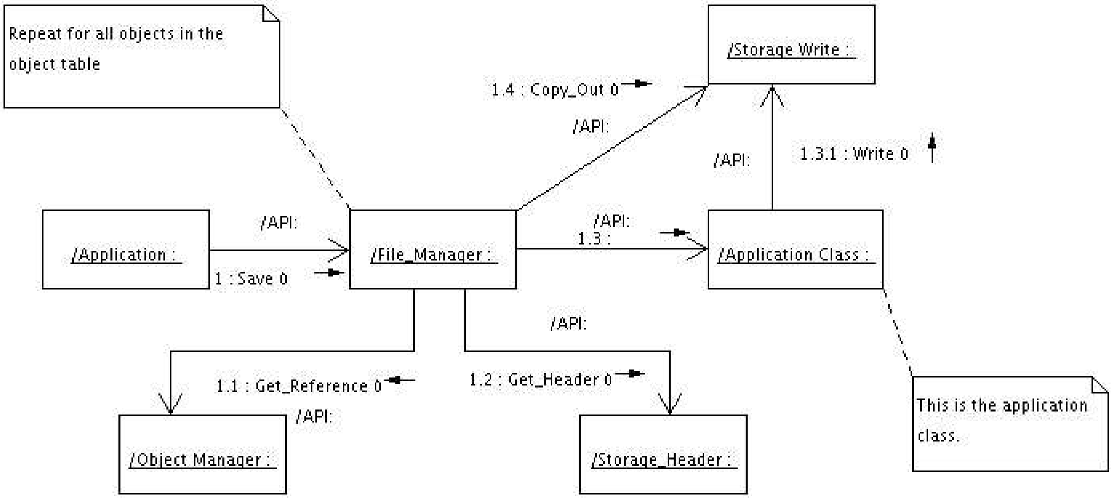
During startup of the application the package will register the attribute names and the corresponding id by the following code fragment:

```
    Class_Id := Classes.Register_Factory( Object'Tag, Factory'Access );

    Classes.Attribute( Class_Id, "D_Name",  D_Name  );
    Classes.Attribute( Class_Id, "D_Used",  D_Used  );
    Classes.Attribute( Class_Id, "D_Pairs", D_Pairs );
```

Storing Objects

When an application decides to terminate it self, the application may decide to store all persistent objects into a persistent storage media by calling the procedure Save.

When calling the procedure Save (e.g. from the component ODB.Storage.File), all named objects are stored on a permanent storage media. This is done by running through a table which contains all persistent information.



The sequence of calls when saving a object to the object storage.

Objects are written by means of the Serialize procedure into a temporary work space, from where the complete object written out into a storage media.

```
procedure Serialize(
   Item   : in out Object;
   Header : in Storage_Header.Handle;
   S      : in Stream_Access ) is abstract;
```

The purpose of this procedure is to write the contents of the attributes into the object storage and the storing the offset of each attribute in the storage header information of the object.

```
procedure Serialize(
    Item   : in out Object;
    Header : in Storage_Header.Handle;
    S      : in Stream_IO.Stream_Access ) is
 begin
    Register_Attribute( Header.all, D_Street, Write_Offset( S ), Object'Tag );
    Unbounded_String'Output( S, Item.Street );

    Register_Attribute( Header.all, D_Name, Write_Offset( S ), Object'Tag );
    Unbounded_String'Output( S, Item.Name );

    .......
 end Serialize;
```

In order to simplify the development, the odl translator generates automatically such procedures.

Reading/Writing Objects

As already mentioned previously the implementation of the read and write procedures have to be symetric, which means what has been written by the Searialize procedure has to to be readable by the Deserialize procedures. Besides of this fact, there are some basic rules to be followed:

- References to other objects can only be stored as references to objects. ODB.Persistent provide a Read/Write method for this type and will resolve the references to other objects in the object store automatically.
- Dynamic data structures have to be resolved by the object implementation, e.g. as in the previous example the array of pairs R.Pairs.
- Any access types in the object have to be resolved by the object implementation (e.g. the ODB.Collection. class)
- For reading and writing use always the operations Input/Output.

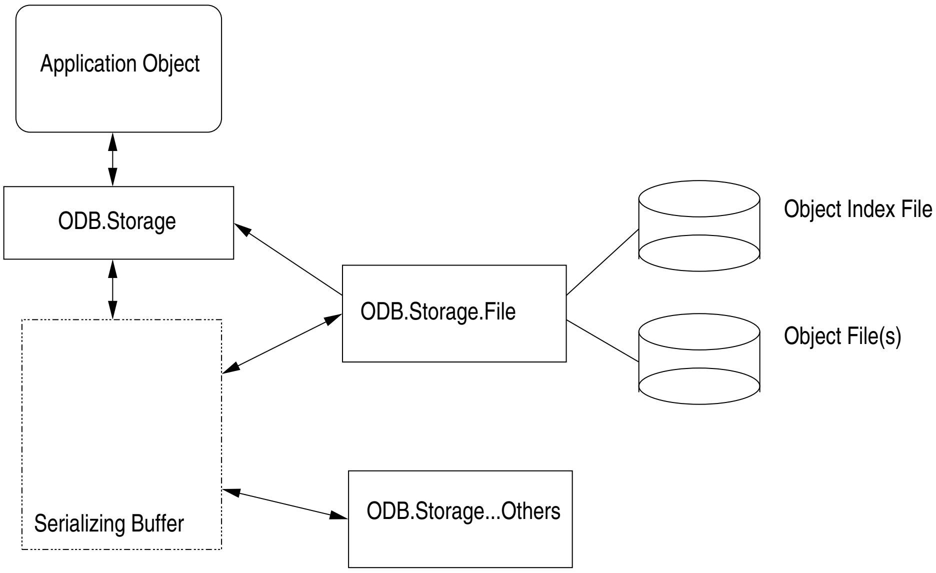
Since the ODL translator is available under normal circumstance the implementation of the read/write procedure by hand is not nessescary since the ODL translator creates the code is self.

ODB API

ODB provides an API to handle persistent objects. This API allowes to Name, lookup and delete objects. The detailed description can be found in the annex of this document

Connection with the storage media

Objects are always serialized/deserialized into a memory buffer (see ODB.Storage). Depending on the storage strategy the serialization buffer will be transferred to or from the target media.



Connection with the storage media

The media specific implementation defined the order in which objects are retrieved from the object table in ODB.Persistent.

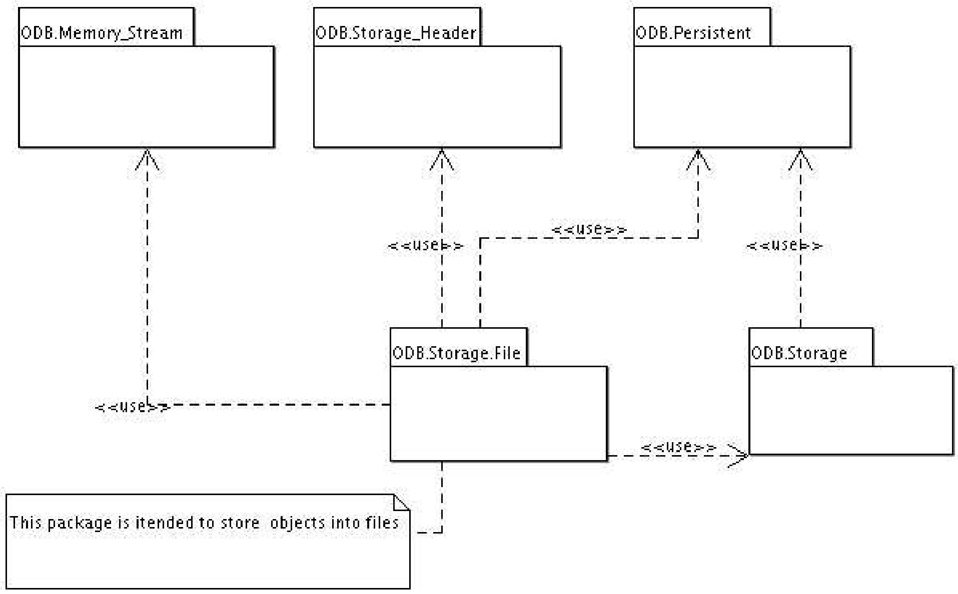
The ODB.Storage.File package retrieves all objects from the object table in sequence and writes the object names into a so called index file and each object is written an individual file.

The individual objects are stored in single files.

Chapter 4. Implemenation

Package Structure

The figure below shows the package structure of the ODB software. The application need to provide at least two packages. One package which contains the persistent class and the application package which uses the persistent class in some sense.



Package Structure of ODB

ODB.Persistent

This package contains the implementation of a storage pool which is used to allocate the memory of persistent objects. Within this package a table is maintained which contains references to all persistent objects ever allocated during the life time of the application.

ODB.Storage

This package implements the basic strategies to save or retrieve an object.

ODB.Storage.File

This package implements the strategy to store the object into the file system.

ODB.Memory_Stream

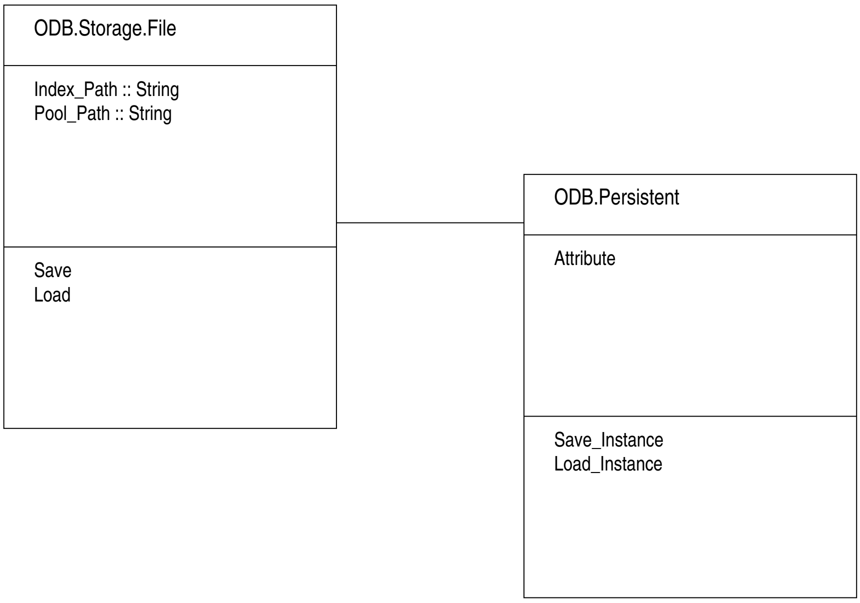
This package allows to read/write via a stream from/into a memory buffer. Is provides additonaly to the normal stream feature operations which are need to navigate directly in the storage buffer of the stream.

ODB.Storage_Header

The data stored for each object is based on two parts the so called storage header and the data it self. The data part is handled by the ODB.Storage package. The header information stored in the storage header contains for each attribute of the object an offset in the data storage where the attribute begins. This information is build up during serialization of the object.

Classes

Since classes in Ada 95 line up with packages each package from the previous chapter represents a class as shown below:



Class Hirarchy

Important to understand is that the class ODB.Persistent is an abstract interface which has to be implemented by persistent objects.

II. User Guide to ODB

Chapter 5. Building Applications

Building an application using persistent objects requires to defined the objects to be handled as persistent objects and to add some basic glue code to your application as shown below:

Example 5-1. A minimal Application

```
with MyObject;
with ODB.Persistent;
with ODB.Storage.File;              ......
use  ODB;
procedure Main is
   O          : ODB.Reference ;
   File_Store : File.Object;
begin
   File.Load(File_Store);

   O := Lookup_Object( "First_Object" );
   if O = null then
      O := new MyObject.Object;
      Object_Name( O, "First_Object" );

      O.A := 1;
   end if;
   ........... do some thing ......

   File.Save(File_Store);
exception
   when Others =>
      ,.......
end Main;
```

In order to use persistency for certain objects you need to define these objects in separate packages. This may be done completely by using Ada 95 or by means of the object definition language (ODL) which is an extension of Ada 95. In the example above, this has been done in the package MyObject which provides a persistent data type Object.

Upon start of the application, it will be checked if the object is already known in the persistency store with the name "First_Object". If not, the object will be allocated by means of the new method.

Chapter 6. Modelling Objects

Since ODL is only a minor extension of Ada 95 only the extensions are described below on basis of examples.

The example below (Test.odb) shows a short fragment. It defines the two types X, Object. The type object is defined as a persistent object which is defined in more detail in the private section of the file.

Example 6-1. A minimal Object Model file

```
package Test  is

   type Object is persistent private;

   type X is new Integer;

private

   type Object is persistent record
          A,X,Y   : attribute Integer ;                 -- this goes into the DB
          B       : attribute String(1..50) ;           -- this also
          C       : Integer := 2222;                    -- this not
       end record ;

end Test;
```

Each field of Object can be marked with the keyword attribute. This keyword indicates, that these parameters will be made persistent. All other field will stay transient. Initializers for all fields may be used but they will only be invoked when the object is first created, with the exception, that fields, which are not attributes, will be set to these values when the application loads the objects.

Definition of persistent Objects

This section describes how persistent object are defined in the ODL language.

Object definition clause

This keyword may be used in a type definition of a record. The keyword indicates, that the following record definition defines a persistent object.

Example 6-2. Syntax: persistent object definition

```
     'type' <name> 'is' 'persistent'
           'record' <attributes;> 'end' 'record'
```

Attribute Definition Clause

The attribute keyword may only be used within persistent object. Itr allows to indicate those fields which will be stored in the persistent object storage.

Example 6-3. Syntax: Attribute Definition

```
attribue def; ::= <field> <attribute def>
field ::= <name list> ':' [ <attribute> ] <type def>
```

All fields in a persistent object which are not marked as an attribute will not be stored in the object storage.

Deriving persistent objects

In order to support inheritance the ODL allows to derive new persistent types from a persistent type.

ISA Clause

The attribute keyword may only be used within persistent object. Itr allows to indicate those fields which will be stored in the persistent object storage.

Example 6-4. Syntax: ISA Clause

```
type <name> ISA <name> with ........... ;
```

Chapter 7. ODB Tools

Since object persistency leads to an extension of the Ada language, the ODB provides several tools to manipulate source codes and other files, which are described in this chapter.

ODL Translator

The ODL Translator is a preprocessor for the input sources of the object model. The object model is always described by the object definition and the corresponding code. These two parts are contained in different files. Assume for example a object Person, then the object model is in the file Person.ods and the code of the object is located in Person.odb. The ODL Translator assumes, that these files are always available. Out of these files, the ODL translator will produce code which complies to the ODL object model containing all procedures and functions which are required by ODB.Persistent to be implemented.

odl [-nognatos] [-s] *name*...

Table 7-1. Options

-gnatnosref	Inserts no sref pragma statement in the output code of the translator.
-s	Run silent, which means no copyright notice etc.

Reserved Words and Names

Since ODL is a preprocessor it will generate code. The application code should not use any name within the name space of ODL.

Limitations

Since the ODL translator is a preprocessor for your Ada compiler the line numbers given in the Ada 95 compiler output are not always correct.

Object Inspector

odlls [-l] *name*...

Table 7-2. Options

-l	Inserts no sref pragma statement in the output code of the translator.

III. API References

Chapter 8. ODB.Attribute_Dictionary

Overview

Functional Description

This package provides a dictionary of attribute/value pairs.

Restrictions

References ========== None

API Reference

procedure Add, function Get, procedure Clear

```
procedure Add(
   This  : in out Object;
   Name  : in String;
   Value : in String );

function Get(
   This  : in Object;
   Name  : in String ) return String;

procedure Clear(
   This  : in out Object );
```

Chapter 9. ODB.Classes

Overview

Functional Description

This package maintains a list of all classes (packages) which are derived from the persistent type ODB.Persistent.Object if the package has registered it self.

The following information is available per class; - Pointer to the so called factory

- All attribute names of a class.

If the application shall support persistence, the following pieces of code has to be added some where during the initialization.

Class_ID := Classes.Register_Factory(Object'Tag, Factory'Access);

Classes.Attribute(Class_ID, "D_Name", D_Name); Classes.Attribute(Class_ID, "D_Used", D_Used); Classes.Attribute(Class_ID, "D_Elements", D_Elements);

This example registers the funtion Factory with Object'tag and adds the attributes D_Name, D_Used and D_Elements.

The factor function creates an instance of the class for which is the function has been registered.

Restrictions

In order to serialize/deserialize the object data, for each attribute of a persistent object a name has to be defined. This is done by means of the Attribute functions (for a typlica example refer to ODB.Collection).

Restrictions

R.1 -

References

None

API Reference

type Factory_Access is access function return Reference

```
type Factory_Access is access function return Reference ;
```

```
Invalid_Attribute_ID : exception;
```

this pointer referes to the Factory function belonging to a certain class.

function Register_Factory

```
function Register_Factory(
   Name    : in Tag;
   Creator : in Factory_Access ) return Natural;
```

Description

Register the factory with the given class name.

Preconditions

- Class is not yet registered

Postconditions

- Factory will return a valid value

Exceptions

None

function Factory

```
function Factory(
   Name : in Tag ) return Factory_Access;
```

Description

Find the factory and return the pointer to the factory function.

Preconditions

- Factory has been registered for the given given persistent class

Postconditions

None

Exceptions

None

function Attribute

```
function Attribute(
   This   : in Tag;
   Id     : in Natural ) return String ;
```

Description

Returns the attribute name for a given attribute id.

Preconditions

- Function returns attribute name

Postconditions

None

Exceptions

None

procedure Attribute

```
   procedure Attribute(
      This : in Natural;
      Name : in String;
      Id   : in Natural );
```

Description

Associate an attribute name and a id with the class id as it has been returned by Register_Factory.

Preconditions

- Factory has been registered for the given given persistent class - Attrbiute has not been defined before.

Postconditions

None

Exceptions

None

function Attribute

```
   function Attribute(
      This   : in Tag;
      Name   : in String ) return Natural;

end ODB.Classes;
```

Description

Translate a attribute name into a attribute id.

Preconditions

- Package class has been registered

Postconditions

- Return 0 if not found

- Returns id if the attribute has been associated with Attute.

Exceptions

None

Chapter 10. ODB.Database

Overview

Functional Description

This object represents the connection to a database.

Restrictions

R.1 -

References

None

API Reference

function Open

```
function Open(
   Arguments : in String ) return Handle is abstract;
```

Description

Preconditions:

Postconditions

Exceptions:

Notes

None

Chapter 11. ODB.Entity

Overview

Functional Description

This package provides entities. Entities do have names and attributes. Attributes are pairs of names and references to other objects.

Restrictions

References ========== None

API Reference

function Create

```
function Create(
   Name : in String ) return Reference;
```

this type represents the persistent collection.

```
Unknown_Attribute : exception;
```

Description

Create an entity of the given name.

Preconditions

None.

Postconditions

P.1 - The function returns a reference to the named entity. If the

named entity already exisits, the reference will be returned. Otherwise a new instance will be created.

Exceptions

None

function Name

```
function Name(
```

```
      This : in Reference ) return String ;
```

Description

Preconditions:

Postconditions

Exceptions:

function Attributes

```
   function Attributes(
      This : in Reference ) return Attribute_Array_Access ;
```

Description

Preconditions:

Postconditions

Exceptions:

procedure Destroy

```
   procedure Destroy(
      This : in out Reference );
```

Description

Desctroy the entity

Preconditions

P.1 - The reference point to a collection.

Postconditions

C.1 - All resources are released

C.2 - The reference to the entity is set to null

Exceptions

Invalid_Object - P.1 violated

function Attribute

```
   function Attribute(
      This   : in Reference;
      Name   : in String ) return Reference;
```

Description

Preconditions:

Postconditions

Exceptions:

procedure Attribute

```
procedure Attribute(
   This   : in Reference;
   Name   : in String;
   Value  : in Reference );
```

Description

Preconditions:

Postconditions

Exceptions:

Chapter 12. ODB.Memory_Stream

Overview

Functional Description

This package contains all defintions needed f7or the linux operating system. This may have to be adopted for other environments.

Restrictions

Only Linux

Contact

Error reports and suggestions shall be send to the Address:

- Michael.Erdmann@snafu.de

purl:/net/michael.erdmann

API Reference

function Stream

```
function Stream(
   Size : in Stream_Element_Offset ) return Stream_Access;


Buffer_Overrun  : exception;
Buffer_Underrun : exception;
```

Description

Create a memory stream

Preconditions

Postconditions:

Exceptions

Notes:

procedure Destroy

```
procedure Destroy(
   This : in out Stream_Access );
```

Description

Destroy the memory stream

Preconditions

C.1 - The stream has to be valid

Postconditions

Exceptions:

procedure Clear

```
procedure Clear(
   This : in Stream_Access );
```

Description

Clear the memory stream.

Preconditions

C.1 - Stream has to be valid

Postconditions

P.1 - Read and write pointer are reseted.

Exceptions

Notes:

function Write_Offset

```
function Write_Offset(
   This : in Stream_Access ) return Natural;
```

Description

Preconditions:

Postconditions

Exceptions:

procedure Read_Offset

```
procedure Read_Offset(
   This   : in Stream_Access;
   Offset : in Natural ) ;
```

Description

Preconditions:

Postconditions

Exceptions:

procedure Copy_In

```
procedure Copy_In(
   This   : in Stream_Access;
   Source : in Stream_Element_Array );
```

Description

Preconditions:

Postconditions

Exceptions:

procedure Copy_Out, function Size

```
procedure Copy_Out(
   This   : in Stream_Access;
   Target : in out Stream_Element_Array;
   Last   : out Stream_Element_Offset );
```

```
function Size(
   This   : in Stream_Access ) return Natural;
```

Description

Preconditions:

Postconditions

Exceptions:

Chapter 13. ODB.Object_Loader

Overview

Functional Description

This package allows to read in an XML document and to read the contents of the document sequentially.

Restrictions

References ========== None

API Reference

procedure Read, function Header, function Stream, function InstanceOf, procedure Path

```
procedure Read(
   This  : in out Object;
   Name  : in String );

function Header(
   This : in Object ) return Storage_Header.Object;

function Stream(
   This : in Object ) return Stream_Access ;

function InstanceOf(
   This : in Object ) return String;

procedure Path(
   This   : in out Object;
   Value  : in String );


Invalid_Object : exception ;
```

Description

Read the named object from the application pool

Preconditions

C.1 - The stream has to be valid

Postconditions

Exceptions:

Chapter 14. ODB.Object_Writer

Overview

Functional Description

Restrictions ============

References

None

API Reference

procedure Path, procedure Write

```
procedure Path(
   This   : in out Object;
   Value  : in String );

procedure Write(
   This   : in out Object;
   Name   : in String;
   Header : in Storage_Header.Object;
   OData  : in Stream_Element_Array );
```

Chapter 15. ODB.Persistent

Overview

Functional Description

This package provides an persistent object storage. All object expected to be persistent have to be derived from the Persistent.Object type. If such an object is to be allocated always use the Reference type Only this access type is associated with the persistent storage pool.

Restrictions

R.1 - This package is not designed to be task save R.2 - The object storage cannot be shared between applications.

References

None

API Reference

procedure Write_Reference, function Read_Reference

```
procedure Write_Reference(
   Stream : access Root_Stream_Type'Class;
   Item   : in Reference );
for Reference'Output use Write_Reference;

function Read_Reference(
   Stream : access Root_Stream_Type'Class ) return Reference;
for Reference'Input use Read_Reference;
```

this type is the root type of all persistent objects use this as references to persistent types

the following two procedures should be ignored, since they are required internaly to read and write references in the object store. They should never be called directly, only by means of the Input, Output attributes.

procedure Serialize

```
procedure Serialize(
   Item   : in out Object;
```

```
      Header : in out Storage_Header.Object;
      S      : in Stream_Access ) is abstract;
```

This execption indicates that an object reference cannot be resolved.

```
   Unresolved_Reference : exception ;
   Unknown_Attribute    : exception ;
```

Maximum number of objects in the Persistent pool.

```
   Max_Nbr_Of_Objects   : constant Natural := 32_000;
```

Description

This procedure writes out the object into the stream provided by the Persistency module. Each field of the object may be written out by means of the Output attribute. References to other objects are automatically expanded into a logical representation which can be read in later if the Method Reference’Output is used.

Preconditions

None

Postconditions

None

Exceptions

None

Notes

Please note, that the application delveloper has to provide this implemenation, but it should never be called somewhere in the application.

This procedure is only called by the persistency manager and should be placed in the private section.

procedure Deserialize

```
   procedure Deserialize(
      Item   : in out Object;
      Header : in out Storage_Header.Object;
      S      : in Stream_Access ) is abstract;
```

Description

Read in all field of the given object. This procedure is called by the persistency manager when loading the objects from an external file.

The procedure may read each field of the object by means of the Input attribute except for references to other instances in the object space. These entires have to be loaded by means of the Reference’Input method.

Preconditions

None

Postconditions

It is assumed, that all fields have been read.

Exceptions

Notes:

Please ensure, that the number of field written is identical to the number of fields read in. If not the file becomes unreadable. This procedure is only called by the persistency manager and should be placed in the private section.

function Factory

```
function Factory return Reference is abstract;
```

Description

The factory funtion creates a new object of the implenting class. Under normal circumstances this will be an new Object operation. This function will be registered together with the External_Tag of the implementation in order to allow the persistency manager to create instances when reading in the data from a file.

Preconditions

None

Postconditions

The function returns a reference to the newly created object

Exceptions

Notes:

Please be aware, that any initialization done in this procedure will be overwritten later when the actual object is restored.

function Object_Id

```
function Object_Id(
   Ref   : in Reference ) return Natural;
```

Description

Returns the object identifier

Preconditions

None

Postconditions

Returns the object identifier which is a Natural number.

Exceptions

Notes:

procedure Name_Object

```
procedure Name_Object(
   Ref  : in Reference ;
   Name : in String );
```

Description

Assign a unique name to the object

Preconditions

C.1 - The Reference has to be valid C.2 - Name has to be unique.

Postconditions

Exceptions:

Invalid_Object - violation of C.1 Duplicate_Name - violation of C.2

function Object_Name

```
function Object_Name(
   Ref  : in Reference ) return String;
```

Description

Ask for the object name

Preconditions

P.1 - Reference is valid

Postconditions

Returns a string with the name of the object

Exceptions

None

function Lookup_Object

```
function Lookup_Object(
   Name : in String ) return Reference;
```

Description

Lookup the object from the object table.

Preconditions

Postconditions:

Returns the reference if the object does exist. Returns null if the object does not exist.

Exceptions

None

function Is_Persistent

```
function Is_Persistent(
   Ref   : in Reference ) return Boolean;
```

Description

Check if the referenced object is persistent

Preconditions

C.1 - The Reference has to be valid

Postconditions

- The function returns true, if the object is persistent.

Exceptions

Invalid_Object - violation of C.1

function Get_Reference

```
function Get_Reference(
   Id    : in Natural;
   Force : Boolean := False ) return Reference;
```

Description

Return the reference to an object from the object identifier

Preconditions

None

Postconditions

Returns the reference to the object if it exists Returns null, if the object does not exist.

Exceptions

None

function Nbr_Of_Objects

```
function Nbr_Of_Objects return Natural;
```

Description

Number of objects in the persistent pool

Preconditions

None

Postconditions

- Returns the number of already stored objects in the pool.

Exceptions

None

Chapter 16. ODB.Storage.File

Overview

Functional Description

This package provides the environment to save persistent objects in a file system. The storage is based upon two files:

The pool index file.

The object pool directory.

The path for pool index and object may be different.

The index file contains a list of all objects which are persistent.

Each individual object is stored as a file using the assigned name in a directory which denoted by the Pool_Path attribute.

Restrictions

R.1 - This package is not designed to be task save R.2 - The object storage cannot be shared between applications.

References

None

API Reference

procedure Pool_Path

```
procedure Pool_Path(
   This  : in out Object;
   Value : String );
```

Description

Set the pool path. This path will be used to locate the place where the object files are stored.

Preconditions

None

Postconditions

None

Exceptions

None

Notes

None

procedure Index_Path

```
procedure Index_Path(
   This  : in out Object;
   Value : String );
```

Description

Preconditions:

- None Postconditions: None Exceptions: None Notes: None

procedure Load

```
procedure Load(
   This  : in out Object );
```

Description

Preconditions:

- None Postconditions: None Exceptions: None Notes: None

procedure Save

```
procedure Save(
   This : in out Object );
```

Description

Preconditions:

- None Postconditions: None Exceptions: None Notes: None

Chapter 17. ODB.Storage

Overview

Functional Description

Restrictions ============

References

None

API Reference

procedure Save_Instance

```
procedure Save_Instance(
   This  : in out Object’Class;
   Item  : in Reference );
```

Description

Save the object in the storage media

Preconditions

Postconditions:

Exceptions

Notes:

procedure Load_Instance

```
procedure Load_Instance(
   This   : in out Object’Class;
   Name   : in String;
   Cls    : in String;
   Phase  : in Load_Phase_Type );
```

Description

Load a named object from the storage media

Preconditions

Postconditions:

Exceptions

Notes:

Chapter 18. ODB.Storage_Header

Overview

Functional Description

Each object is described by a so called Storage_Header. The storage- header contains the names of each field which has been declared as an attrbute in the ODL specification.

+-------------------+

| Object Class Name |

+-------------------+

| D_Name | offset |------+ Object Header +-------------------+ |

| D_Street | offset |---+ |

: : | |

| | | |

+-------------------+ .... | Michael Erdmann |<--+ |

| Some where |<-----+

+-------------------+

This package provides methods to manipulate the contents of the Object header.

Restrictions

References ========== None

API Reference

procedure Register_Attribute, procedure Register_Attribute

```
procedure Register_Attribute(
   This   : in out Object;
   Name   : in Unbounded_String;
   Offset : in Natural );

procedure Register_Attribute(
   This   : in out Object;
   Id     : in Natural;
```

```
      Offset : in Natural;
      Cls    : in Tag );


   Unknown_Attribute : exception;
```

Description

Resgister the attrbute based on the field id as it is registered in the class it self.

Preconditions

C.1 - Object is valid.

C.2 - Tag references an object derived from ODB.Persistent.Object

Postconditions

Exceptions:

function Lookup_Attribute

```
   function Lookup_Attribute(
      This   : in Object;
      Name   : in String ) return Natural;
```

Description

Return the offset for the named attribute in the header.

Preconditions

C.1 - The attribute Name has been registered via Register_Attribute

Postconditions

P.1 - Returns offset

Exceptions

Unknown_Attribute : C.1 violated

function Attributes

```
   function Attributes(
      This   : in Object ) return String_Array.Handle;
```

Description

Return all attribute names which are registered in the header.

Preconditions

C.1 - Object is valid

Postconditions

P.1 - Returns null if the header is empty P.2 - Returns the point to a string array.

Exceptions

Notes:

The string array has to be destroyed by means of the operation Free in Util.String_Array.

procedure Clear, function Class_Name, procedure Class_Name

```
procedure Clear(
   This   : in out Object );

function Class_Name(
   This   : in Object ) return String ;

procedure Class_Name(
   This   : in out Object;
   Value  : in String );
```

Description

Clear the header.

Preconditions

C.1 - Object is valid

Postconditions

P.1 - All attributes are removed from the header.

Exceptions

Notes:

Chapter 19. ODB.Transaction

Overview

Functional Description

This package implements a elementary transaction system by providing the cpability of locking an instance.

At the begon of each transation, a copy of the original persistent objec will be created and the object is locked for other transactions. From this point on, the owner process of the transaction may manipulate the contents of the object.

Be aware, since every process may retrieve at any time the object reference from the persistency manager it is possible that a thread may see intermediate results unless the implementation starts first a transaction.

Restrictions

References ========== None

API Reference

procedure Initialize

```
procedure Initialize(
   Size : in Natural );


Invalid_Transaction : exception;
Invalid_Usage       : exception;
```

Description

Initialize the transaction manager package to handle the given number of transactions.

Preconditions

P.1 - Transaction Manager not initialized.

Postconditions

C.1 - The object value is the same as at the time of the invokation

of the start method.

C.2 - Transaction is still active.

Exceptions

P.1 - Unvalid_Use

procedure Finalize;

```
procedure Finalize;
```

Description

Shuttdown the transaction management

Preconditions

P.1 - Transaction manager is Initialized.

Postconditions

C.1 - All tranaction data is lost. C.2 - Transaction Monitoring is stoped.

Exceptions

Notes:

procedure Start

```
procedure Start(
   This     : in out Object;
   Instance : in Persistent.Reference );
```

Description

Start a transaction. This procedure blocks if other processes do have an active transaction. The transaction is closed either by the Commit of the Cancel method.

Preconditions

P.1 - The transaction package has been initialized by means of the

initialize procedure.

Postconditions

C.1 - A copy of the given persistent object is created. C.2 - The object is locked for other transactions. The method will

block until the instance has been aquiered by the process.

Exceptions

Notes:

procedure Commit

```
procedure Commit(
   This : in out Object );
```

Description

Close the transaction.

Preconditions

P.1 - Transaction has been Started

Postconditions

C.1 - The backup copy of the object is deleted and the object is

available for other processes to aquiere the object by means of a start operation.

Exceptions

Notes:

procedure Cancel

```
procedure Cancel(
   This : in out Object );
```

Description

Cancel the current transaction.

Preconditions

P.1 - Transction has been started

Postconditions

C.1 - The Original value of the object is restored C.2 - object is available for other transactions.

Exceptions

Notes:

procedure Rollback

```
procedure Rollback(
   This : in out Object );
```

Description

Restore the contents of the object to the original value

Preconditions

P.1 - Transaction is active

Postconditions

C.1 - The object value is the same as at the time of the invokation of the start method.

C.2 - Transaction is still active.

Exceptions

Notes:

Chapter 20. ODB.XML

Overview

Functional Description

This package contains the XML tag names and the name space specification Changes to the tag names should only be done here in order to keep the Loader and the Writer symetrical.

Restrictions

Syntax changes need to be done in the Object Loader and Writer.

References

None

Chapter 21. ODB

Overview

Functional Description

This is the top level package of the ODB package hirarchy. It contains some basic definitions valid to the complete source tree as for example exceptions.

Restrictions

References ========== None

Chapter 22. Util.Hash_Table

Overview

Functional Description

This package implements an container which stores pairs of objects. derived from the Keys and the Container_Element package. Each object derived from the Keys package is called a key. This package allows to retrieve (address) the stored Container_Element by means of the key object.

The Keyed container has a maximum capacity for such pairs.

Upon storing the container creates a complete copy of the <key,object> pair in order to ensure that there are no references from outside into sub components of the stored objects.

Restrictions

R.1 - The handling of the tree nodes is currently not task save

References

None

API Reference

Generic Package Parameter(s)

```
type Key_Type is private ;
```

procedure Put

```
procedure Put(
    This   : in out Table_Type;
    Key    : in Key_Type;
    Value  : out Natural );


Table_Full        : exception;
Key_Already_Used  : exception;
Key_Not_Existing  : exception;
```

Description

This method stores the key and the container element in the keyed container.

Preconditions

P.1 - Item /= null

P.2 - Number of stored keys < capacity.

P.3 - Key not already stored

Postconditions

C.1 - An additional key/value pair has been added to the container.

Exceptions

Table_Full - P.2 violated

function Get

```
function Get(
    This : in Table_Type;
    Key  : in Key_Type ) return Natural;
```

Description

Retrieve a container element from the container.

Preconditions

P.1 - Key exisits in container

Postconditions

C.1 - Funtion yield handle to copy of the container element

Exceptions

Key_Not_Existing - P.1 violated

Notes

None

function Key

```
function Key(
    This : in Table_Type;
    Hash : in Natural ) return Key_Type;
```

Description

Get the key associated with the hash code

Preconditions

P.1 - Hash code is used

Postconditions

C.1 - The yields key

Exceptions

Key_Not_Existing - P.1 violated

Notes

None

procedure Remove

```
procedure Remove(
   This : in out Table_Type;
   Key  : in Key_Type );
```

Description

Remove a key from the container

Preconditions

P.1 - Key exists in container

Postconditions

C.1 - The container_element held by the container is deallocated

Exceptions

Key_Not_Existing - P.1 violated

procedure Clear

```
procedure Clear(
    This : in out Table_Type);
```

Description

Clear the container

Preconditions

None

Postconditions

C.1 - All objects held by the container are destroyed. The

nbr of free entries = capacity.

Exceptions

None

Notes

None

function Is_Empty

```
function Is_Empty(
   This    : in Table_Type) return Boolean;
```

Description

Check if empty

Preconditions

None

Postconditions

C.1 - return true if nbr_of_entries = 0

Exceptions

None

function Contains

```
function Contains(
   This    : in Table_Type;
   Key     : in Key_Type ) return Boolean;
```

Description

check if key or a element is in the container

Preconditions

None

Postconditions

C.1 - either true or false or the key handle.

Exceptions

None

Notes

Both procedures are not optimized for speed. In case of large tables this might be a real problem for the performance of your application code.

procedure Dictionary

```
procedure Dictionary(
   This    : in  Table_Type;
   Keys    : out Dictionary_Table;
   Length  : out Natural );

Dictionary_Overflow : exception;
```

Description

Returns the list of all keys in the container

Preconditions

P.1 - The number of entries in the container is not larger then

the provided dictionary table

Postconditions

C.1 - length is set to the actual number of entries in the

dictionary.

Exceptions

Dictionary_Overflow - P.1 violated.

Notes

None

Chapter 23. UTIL.List

Overview

Functional Description

This generic package provides a basic list for a given data type.

Restrictions

References ========== None

API Reference

Generic Package Parameter(s)

```
type Item_Type is private;
```

function List

```
function List return Handle;
```

Description

Create a list.

Preconditions

Postconditions:

The function returns a list handle.

Exceptions

Notes:

procedure Destroy

```
procedure Destroy(
   This : in out Handle );
```

Description

Destroy the list

Preconditions

C.1 - The list handle is valid.

Postconditions

All allocated resources are returned.

Exceptions

Notes:

procedure Append

```
procedure Append(
   This : in Handle;
   Item : in Item_Type;
   Sub  : in Handle := Null_Handle );
```

Description

Append an element to the given list handle

Preconditions

C.1 - The list Handle has to be valid.

Postconditions

Exceptions:

function Length

```
function Length(
   This : in Handle ) return Natural;
```

Description

Get the length of the list

Preconditions

- C.1 List is valid.

Postconditions

- P.1 Nothing changed

- P.2 Returns the length of the list. If the list is empty a 0 is returned.

Exceptions

Notes:

procedure Execute, procedure Execute, procedure Stop, type List_Reader_Handle

```
procedure Execute(
   It      : in out Iterator;
   Element : in out Item_Type ) is abstract;

procedure Execute(
   This : in Handle;
   It   : in out Iterator'Class );

procedure Stop(
   It   : in out Iterator'Class );

type List_Reader_Handle is private;
Null_List_Reader_Handle : constant List_Reader_Handle ;
```

Description

Append an element to the given list handle

Preconditions

Postconditions:

Exceptions

Notes:

function List_Reader

```
function List_Reader(
   This : in Handle ) return List_Reader_Handle;
```

Description

Create a list reader for the given list

Preconditions

C.1 - List is valid

Postconditions

P.1 - The list reader points to the begin of the list.

Exceptions

Notes:

In order to loop through a list, use the First function to obtain the first element in the list.

procedure Destroy, function Child , procedure Child

```
procedure Destroy(
   Reader : in out List_Reader_Handle );

End_Of_List  : exception ;
Invalid_List : exception ;

function Child (
   Reader : in List_Reader_Handle ) return Handle;

procedure Child(
   Reader : in List_Reader_Handle;
   List   : in Handle );
```

Description

Dstroy the list reader

Preconditions

C.1 - Listreader is valid

Postconditions

P.1 - All allocated resources are deallocated.

Exceptions

Notes:

function First

```
function First(
   Reader : in List_Reader_Handle ) return Item_Type;
```

Description

Set the reader on the first element

Preconditions

C.1 - Listreader is valid

Postconditions

P.1 - the function returns the first element

Exceptions

Notes:

function Next

```
function Next(
   Reader : in List_Reader_Handle ) return Item_Type;
```

Description

Set the reader on the first element

Preconditions

C.1 - Listreader is valid

Postconditions

P.1 - the function returns the first element

Exceptions

Notes:

procedure Append

```
procedure Append(
   Reader : in List_Reader_Handle;
   Data   : in Item_Type );
```

Description

Preconditions:

C.1 - Listreader is valid

Postconditions

P.1 - the function returns the first element

Exceptions

Notes:

function Current

```
function Current(
   Reader : in List_Reader_Handle ) return Item_Type;
```

Description

Get the item where the current readpoint points to.

Preconditions

C.1 - Listreader is valid

Postconditions

P.1 - Nothing is changed. P.2 - Current element is returned.

Exceptions

Notes:

function List_End

```
function List_End (
   Reader : in List_Reader_Handle ) return Boolean;
```

Description

Check for list end

Preconditions

C.1 - Listreader is valid

Postconditions

P.1 - Returns true if end of list is reached, else false.

Exceptions

Notes:

Chapter 24. Util.Lock_Table

Overview

Functional Description

This package provides a general resource locking functionality. The call need to address each resource by means of a ntural nunber. This identifier will be stored in the so call lock_table object. If a resource is allready seized the seize function block until the resource becomes available.

Restrictions

References ========== None

API Reference

function Seize

```
function Seize(
   This : in Object;
   Id   : in Natural ) return Lock_Handle_Type;


Invalid_Lock_Handle : exception;
```

Description

Seize a resource which is identified by the given natural number. The function blocks till the given resource is available.

Preconditions

Postconditions:

C.1 - Returns a so called lock identifier which has to be used

to unlock the seized resource.

Exceptions

Notes:

procedure Release

```
procedure Release(
```

```
This : in out Object;
Lock : in out Lock_Handle_Type );
```

Description

Make the resource identified by the lock identifier available for other seizures.

Preconditions

P.1 - Lock identifier exisits.

Postconditions

C.1 - The resource is available for other seizures.

Exceptions

Notes:

Chapter 25. UTIL.Stack

Overview

Functional Description

Restrictions ============

References

None

API Reference

Generic Package Parameter(s)

```
type Item_Type is private;
```

function New_Stack, procedure Destroy, procedure Push, procedure Pop, function Current, function Is_Empty

```
function New_Stack return Handle;

procedure Destroy(
   This : in out Handle );

Stack_Empty : exception;

procedure Push(
   This    : in Handle;
   Value   : in Item_Type );

procedure Pop(
   This    : in Handle;
   Value   : in out Item_Type );

function Current(
   This    : in Handle ) return Item_Type ;

function Is_Empty(
   This    : in Handle ) return Boolean;
```

Chapter 26. Util.String_Array

Overview

Functional Description

This is the root package for all utilities of the ODB project. These components are not subject to development effort of this project but they are needed to implement the project. Since they might of some use they are documented here.

Restrictions

============

References

None

API Reference

procedure Free

```
   procedure Free( This : in out Handle );

end Util.String_Array;
```

Chapter 27. Util.String_Map

Overview

Functional Description

Restrictions ============

References

None

API Reference

Generic Package Parameter(s)

```
type Value_Type is private;
```

procedure Put, function Get, function Is_Empty, function Key

```
procedure Put(
   This  : in out Object;
   Key   : in String;
   Value : in Value_Type );


function Get(
   This : in Object;
   Key  : in String ) return Value_Type;


function Is_Empty(
   This : in Object ) return Boolean;

function Key(
   This : in Object;
   H    : in Natural ) return String;
```

Description

Add an Key/Value pair to the string map

Preconditions

C.1 - The key is not already in the table

Postconditions

The Value is stored in the map and may be retrieved by means of the Get method.

Exceptions

Notes:

Chapter 28. Util

Overview

Functional Description

This is the root package for all utilities of the ODB project. These components are not subject to development effort of this project but they are needed to implement the project. Since they might of some use they are documented here.

Restrictions

============

References

None

Appendix A. Frequently asked questions

This section contains the FAQ's of the ABE project.

Q: How do i update

How can i update the ABE environment without disturging any thing?

....

Example A-1. Updating ABE

Appendix B. GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Appendix C. GNU Public License (GPL) Version 2

```
       GNU GENERAL PUBLIC LICENSE
          Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
     59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

        Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation’s software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author’s protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
```

authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then

```
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

       NO WARRANTY
```

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Colophon

Draft versions of this book were produced with the DocBook DSSSL Stylesheets. Final production was performed with LaTex and ps2pdf.

The UML diagrams have been produced using ArgoUML Version 0.14.a4 (see http://argouml.tigris.org).