Mnesia_Session

version 1.1

Typeset in LaTeX from SGML source using the DOCBUILDER 3.3.2 Document System.

Contents

Chapter 1

Mnesia_Session User's Guide

1.1 Introduction

This book describes Mnesia_Session. It is an interface to the Mnesia Database Management System and is a part of the Erlang/OTP, which is a control system platform for building telecommunications applications.

Mnesia is written in Erlang and is intended for use in conjunction with Erlang applications.

The Mnesia_Session application enables access to the Mnesia DBMS from foreign programming languages (i.e. other languages than Erlang). The Mnesia_Session interface is defined in IDL (an Interface Definition Language standardized by OMG (the Object Management Group)). Access is enabled via the following protocols:

- IIOP (Internet Inter ORB Protocol) standardized by OMG.
- the proprietary distribution protocol of Erlang Erl_Interface.

Details of these features are described in the following sections.

1.1.1 Scope and Purpose

This manual is included in the OTP document set. It describes the syntax, but not the semantics, of how Mnesia is accessed via the Mnesia_Session interfaces. Programming constructs are described, and numerous programming examples are included to illustrate the use of Mnesia_Session.

1.1.2 Pre-requisites

Readers of this manual are assumed to be familiar with system development principles and database management systems. Readers are also assumed to be familiar with the Erlang programming language in general and the Mnesia, Orber and IC applications in particular.

1.2 Overview

1.2.1 Interfaces

The interfaces of the *Mnesia_Session* application are defined in IDL (Interface Definition Language). The IDL module `mnesia` consists of two interfaces:

- The *connector* (and possibly corba_connector) is started statically when starting the Mnesia_Session application.
- The *session* (and corba_session) which is started dynamically at the request of a client.

When a Mnesia_Session client needs to access the Mnesia DBMS it locates a `connector` on some Erlang node and starts a `session` there by applying the `connect` function on the connector.

Once the client has started a `session` it may use it to perform various Mnesia functions. The functions are performed locally on the Erlang node where the `session` resides. Most of Mnesia's functions have location transparent behavior, but some of them (e.g. `start/0`) do not.

The Mnesia_Session interfaces makes it possible to access the administration and dirty functionality of Mnesia. To use transactions and/or Mnemosyne queries it is up to the user to define the needed interface in IDL. Implementing a user specified interface allows an opportunity to skip the general handling of `erlang::term`, which can sometimes be troublesome when using foreign languages, for both `mnesia_session` and the `mnesia_corba_session`.

See the IDL specification [page 5] for functions which are available. The specification resembles the Mnesia API as much as possible. Please, read the Mnesia documentation set regarding the semantics used in the interface.

All the functions in the session (and corba_session) API return `Status` which indicates if the operation was successful or not. Most of the functions have an out parameter with a string `reason` describing the error, if one occurs.

> **Note:**
> The return value `Status` should be checked after each call, and it should be matched against the `Status` enum `ok`. For some functions, the `end_of_table` also means that the operation was successful.

1.2.2 Communication protocols

The IDL specification of Mnesia_Session has two alternatives when compiling. These can be found in the `mnesia_session/include` directory:

- mnesia_session.idl
- mnesia_corba_session.idl

The `mnesia_session.idl` file must be compiled with `IC` (OTP's own IDL compiler). The generated stub files use the proprietary distribution protocol of Erlang (erl_interface/jinterface) to carry out the communication between clients and servers of connectors and sessions. On the server side Mnesia_Session is implemented in Erlang using `Mnesia`'s public API. On the client side Erlang, Java or C may be used.

The `mnesia_corba_session.idl` file may be compiled with any Corba compliant IDL compiler (e.g. Orbix, JacORB, TelORB, IC, ...) . The generated stub files uses IIOP (a protocol standardized by OMG) to carry out the communication between clients and servers of connectors and sessions. On the server

side Mnesia_Session is implemented in Erlang using Mnesia's public API. On the client side a wide range of programming languages are available: Java, Smalltalk, C++, Erlang etc.

1.2.3 Sessions

When the Mnesia_Session application is started, an Erlang process with the registered name *mnesia_connector* is created. The following example illustrates how a `session` is started:

```
% erl

1> application:start(mnesia_session).
ok
2> Name = mnesia_connector,
mnesia_connector
3> Connector = erlang:whereis(Name).
<0.34.0>
4> Session = mnesia_connector:connect(Connector).
<0.35.0>
5> ok = mnesia_connector:disconnect(Connector, Session).
ok
```

Note:
In the example given, both the client and server reside (in Erlang) on the same node.

See the Orber and IC documentation about the language mapping between Erlang, Java, C and IDL.

1.2.4 CORBA Sessions

If the Mnesia_Session application has been started with the configuration parameter `enable_corba` set to `true`, a mnesia_corba_connector object is also created (in addition to the mandatory mnesia_connector process), and registered in Orber. The following simplified example illustrates how a `corba_session` can be started:

```
% erl -mnesia_session enable_corba true

1> application:start(mnesia_session).

2> NS = corba:resolve_initial_references("NameService").
3> NC = lname_component:set_id(lname_component:create(),
                               "mnesia_corba_connector").
4> Name = lname:insert_component(lname:create(), 1, NC).
5> Connector = 'CosNaming_NamingContext':resolve(NS, Name).

6> Session = mnesia_corba_connector:connect(Connector).
7> mnesia_corba_connector:disconnect(Connector, Session).
```

Note:
In the example given, both the client and server reside (in Erlang) on the same node.

More information about CORBA conventions and usage can be found in the Orber and IC documentation.

Since Orber uses Mnesia internally, some of the functions in the Mnesia API are not available via IIOP. Examples of such functions are:

- start;
- stop;
- create_schema; and,
- delete_schema.

See the IDL specification [page 5] for the exact specification.

Some other functions are not supported due to the problem of representing void objects of unknown types. The `dirty_[index_]match_object` functionality has been replaced with the simpler function `dirty_match_all` which returns all records in a table.

1.2.5 Erlang, C and Java Sessions

These sessions are faster and more flexible variants than the CORBA session. The protocol implemented by the generated stubs is Erlang native external communication protocol, there is no ORB and IIOP engaged. Due to these facts the clients will run 10 to 20 times faster than in CORBA session case, depending on the amount of data stored and the mnesia record seeking times.

1.2.6 User-defined Interfaces

To be able to send records over the IIOP protocol, the records must be defined as structures in an IDL specification, and compiled with IC in order to enable registering of the types in Orber's InterFace Repository (IFR). The records are mapped to the type `any` in Corba.

We recommend that all records are defined as IDL structures. This also applies when the `erl_interface` protocol is used (even though it may work without it). By including the header files produced in the code generation, several useful type definitions are made available for the application.

The generic dirty access functions in the API of Mnesia_Session is merely included for the convenience of application developers and it may be tempting to organize the application code around these functions. The application interface between its clients and servers, should however be carefully designed according to the needs of the application, regardless of the Mnesia_Session interface.

Instead of sending records back and forth between the server and client nodes as in the generic get-/put-oriented interface of Mnesia_Session, it may (in many cases) be a better application design, to perform the application logic on the same (Erlang) node as the residing data. Besides the obvious performance advantage, it makes the applications more independent of future changes in the data model of the application.

1.3 IDL Specification of Mnesia (CORBA) Session

1.3.1 mnesia_session.idl

```
// ‘‘The contents of this file are subject to the Erlang Public License,
// Version 1.1, (the "License"); you may not use this file except in
// compliance with the License. You should have received a copy of the
// Erlang Public License along with this software. If not, it can be
// retrieved via the world wide web at http://www.erlang.org/.
//
// Software distributed under the License is distributed on an "AS IS"
// basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See
// the License for the specific language governing rights and limitations
// under the License.
//
// The Initial Developer of the Original Code is Ericsson Utvecklings AB.
// Portions created by Ericsson are Copyright 1999, Ericsson Utvecklings
// AB. All Rights Reserved.’’
//
//     $Id$
//
////////////////////////////////////////////////////////////////////
// Mnesia Session - IDL interface to Mnesia
////////////////////////////////////////////////////////////////////

#include <erlang.idl>

// All types are enclosed by the mnesia module

module mnesia
{
  ////////////////////////////////////////////////////////////////
  // Status - a general type for keeping track of
  // the outcome of the session related functions.

  enum Status
  {
    // Success codes:

    ok,            // The operation was successful and
                   // the all out parameters except ’reason’
                   // contains valid data.

    end_of_table,  // End of table has been reached.
                   // This return code is only used by
                   // a few functions, such as dirty_slot/4,
                   // dirty_first/3, dirty_next/4 ...

    // Failure codes:

    timeout,     // This return code is only used by
                 // wait_for_table.
```

```
   error        // Miscellaneous error.
                // Operation did not succeed and the out
                // parameter 'reason' contains a description
                // of the error. The other out parameters
                // contains garbage.

  // In future releases we will probably expand this enum
  // with more precise error codes like no_exists, badarg etc.
 };

 // Record - an erlang record, i.e. a tagged tuple with the
 // following layout: {tab_name, key, attr_1, ... attr_N}

 typedef erlang::term Record;
 typedef sequence <Record> Recordlist;

 // TableName - a name of a table. Note that the character '_'
 // should be avoided if the table is intended to be used
 // via the Corba Session.

 typedef string TableName;
 typedef sequence <TableName> TableList;

 // Key - a key of a record

 typedef erlang::term Key;
 typedef sequence <Key> KeyList;

 // Indecies - list of index positions.
 // In the sample record above, attr_1 has position 3

 typedef sequence <long> Indices;

 // AttrNames - a list of attribute names

 typedef sequence <string> AttrNames;

 // RecordName - Type information

 typedef string RecordName;

 // Node - an Erlang node, e.g. "name1@host[.org.country]"

 typedef string Node;
 typedef sequence <Node> NodeList;

 // Storage - table storage type

 enum Storage {ram_copies, disc_copies, disc_only_copies};

 // SetOrBag - table type
```

```
enum SetOrBag {set, bag};

// AccessMode - controls the accessibility of a table

enum AccessMode {read_only, read_write};

// Checkpoints - a list of checkpoint names

typedef sequence <string> Checkpoints;

//////////////////////////////////////////////////////////////
// TableDef - initial specification of a table

struct TableDef
{
  SetOrBag type;
  AccessMode mode;
  NodeList ram_copies;
  NodeList disc_copies;
  NodeList disc_only_copies;
  Indices index_list;
  AttrNames attributes;
  RecordName record_name;
};

//////////////////////////////////////////////////////////////
// CheckpointDef - initial specification of a checkpoint

struct CheckpointDef
{
  string cpName;
  TableList max;
  TableList min;
  boolean allow_remote;
  boolean ram_overrides_dump;
};

//////////////////////////////////////////////////////////////
// TableInfo - information about a table

struct TableInfo
{
  AccessMode mode;
  AttrNames  attributes;
  long arity;
  Checkpoints checkpoints;
  NodeList ram_copies;
  NodeList disc_copies;
  NodeList disc_only_copies;
  Indices indexlist;
  long load_order;
  boolean local_content;
  NodeList master_nodes;
```

```
  long memory;
  long size;
  Storage storage_type;
  SetOrBag type;
  Node where_to_read;
  NodeList where_to_write;
  RecordName record_name;

  // Not supported: cookie, snmp, subscribers, version, wild_pattern
};

////////////////////////////////////////////////////////////
// SystemInfo - information about the Mnesia system
// Some info can always be determined, but
// other info is only valid if Mnesia is running.

struct SystemInfo
{
  // The following information is always determined by
  // system_info/2:

  boolean auto_repair;
  string backup_module;
  string event_module;
  NodeList db_nodes;
  string debug;
  string directory;
  boolean dump_log_load_regulation;
  long dump_log_time_threshold;
  boolean dump_log_update_in_place;
  long dump_log_write_threshold;
  NodeList extra_db_nodes;
  boolean is_running;
  string schema_location;
  boolean use_dir;
  // Not supported: log_version, protocol_version, version

  // The following information is only valid if Mnesia is running.
  // Check 'is_running' before the info is accessed and regard the
  // values as undefined if 'is_running' is false.

  Checkpoints checkpoints;
  boolean fallback_activated;
  TableList local_tables;
  TableList master_node_tables;
  NodeList running_db_nodes;
  TableList tables;
  long transaction_failures;
  long transaction_commits;
  long transaction_restarts;
  long transaction_log_writes;

  // Not supported: held_locks, lock_queue, communication protocol,
```

```
    //                subscribers, transactions
  };

////////////////////////////////////////////////////////////////
// Most of the following functions returns Status and in case of
// an error, the out parameter 'reason' contains the description
// of the cause and all other out parameters contains undefined values.
//
// If the function was successful, 'reason' contains an empty string
// and the other out parameters contains valid data.

  interface session
    {
      ////////////////////////////////////////////////////////////
      // Dirty access functions

      Status dirty_write(         in  TableName tab,
                                  in  Record object,
                                  out string reason);

      Status dirty_read(          in  TableName tab,
                                  in  Key key,
                                  out Recordlist result,
                                  out string reason);

      Status dirty_update_counter(in  TableName tab,
                                  in  Key key,
                                  in  long val,
                                  out long newval,
                                  out string reason);

      Status dirty_delete(        in  TableName tab,
                                  in  Key key,
                                  out string reason);

      Status dirty_delete_object( in TableName tab,
                                  in  Record object,
                                  out string reason);

      Status dirty_slot(          in  TableName tab,
                                  in  long slot,
                                  out Recordlist result,
                                  out string reason);

      Status dirty_first(         in  TableName tab,
                                  out Key next_key,
                                  out string reason);

      Status dirty_next(          in  TableName tab,
                                  in  Key key,
                                  out Key next_key,
                                  out string reason);
```

```
Status dirty_all_keys(      in  TableName tab,
                            out KeyList keys,
                            out string reason);

Status dirty_match_all(     in  TableName tab,
                            out Recordlist result,
                            out string reason);

Status dirty_index_read(    in  TableName tab,
                            in  Key key,
                            in  long pos,
                            out Recordlist result,
                            out string reason);

////////////////////////////////////////////////////////////
// Table management

Status create_table(            in  TableName tab,
                                in  TableDef tabDefs,
                                out string reason);

Status delete_table(            in  TableName tab,
                                out string reason);

Status add_table_copy(          in  TableName tab,
                                in  Node node,
                                in  Storage type,
                                out string reason);

Status del_table_copy(          in  TableName tab,
                                in  Node node,
                                out string reason);

Status move_table_copy(         in  TableName tab,
                                in  Node from,
                                in  Node to,
                                out string reason);

Status add_table_index(         in  TableName tab,
                                in  long attrname,
                                out string reason);

Status del_table_index(         in  TableName tab,
                                in  long attrname,
                                out string reason);

Status change_table_copy_type(  in  TableName tab,
                                in  Node node,
                                in  Storage type,
                                out string reason);

Status change_table_access_mode(in  TableName tab,
                                in  AccessMode mode,
```

```
                                out string reason);

////////////////////////////////////////////////////////////
// Table load
Status wait_for_tables(        in  TableList tabs,
                               in  long timeout,
                               out TableList failed_tabs,
                               out string reason);

Status force_load_table(       in  TableName tab,
                               out string reason);

Status change_table_load_order(in  TableName tab,
                               in  long load_order,
                               out string reason);

Status set_master_nodes1(      in  NodeList nodes,
                               out string reason);

Status set_master_nodes2(      in  TableName tab,
                               in  NodeList nodes,
                               out string reason);

////////////////////////////////////////////////////////////
// Database management

Status backup1(              in  string filename,
                             out string reason);

Status backup2(              in  erlang::term opaque,
                             in  string modulename,
                             out string reason);

Status install_fallback1(    in  string filename,
                             out string reason);

Status install_fallback2(    in  erlang::term opaque,
                             in  string modulename,
                             out string reason);

Status uninstall_fallback(   out string reason);

Status dump_log(             out string reason);

Status dump_tables(          in  TableList list,
                             out string reason);

Status activate_checkpoint(  in  CheckpointDef cpdef,
                             out string reason);

Status deactivate_checkpoint(in  string cpName,
                             out string reason);
```

```
Status backup_checkpoint1(   in  string cpName,
                             in  string filename,
                             out string reason);

Status backup_checkpoint2(   in  string cpName,
                             in  erlang::term opaque,
                             in  string modulename,
                             out string reason);

////////////////////////////////////////////////////////////
// Miscellaneous

Status load_textfile(   in  string filename,
                        out string reason);

Status dump_to_textfile(in  string filename,
                        out string reason);

Status table_info(      in  TableName tab,
                        out TableInfo info,
                        out string reason);

Status system_info(     out SystemInfo info,
                        out string reason);

////////////////////////////////////////////////////////////
// match_object functions is not supported due to the fact that
// erlang::term is represented as an 'any' object and can not
// be handled in an ordinary way in corba

Status dirty_match_object(      in  TableName Tab,
                                in  Record pattern,
                                out Recordlist result,
                                out string reason);

Status dirty_index_match_object(in TableName Tab,
                                in  Record pattern,
                                in  long pos,
                                out Recordlist result,
                                out string reason);

////////////////////////////////////////////////////////////
// Functions that deals with starting and stopping the Mnesia
// application or operates when Mnesia is not running is not
// supported in the corba interface due to fact that Orber
// needs a running Mnesia in order to operate.

Status create_schema(  in  NodeList nodes,
                       out string reason);

Status delete_schema(  in  NodeList nodes,
                       out string reason);
```

```
      Status    start_mnesia(out string reason);

      Status    stop_mnesia( out string reason);
    };



////////////////////////////////////////////////////////////////
// The connector process registers itself in the local Erlang node
// at startup of the Mnesia Session application

  interface connector
    {
      // Connects to Mnesia on the same Erlang node as the
      // Connector process, creates a Session process and returns
      // its process identifier.

      erlang::pid connect();

      // Disconnects from Mnesia and terminates
      // the Session process.

      void disconnect(in erlang::pid object_key);
    };

};
```

1.4 Session Examples

1.4.1 Example Introduction

In this and the following chapter several examples of the usage of the Mnesia_Session interface will be introduced. The examples will first be written in Erlang and than the same functionality will be shown in a foreign language.

The examples will contain some basic operations, create_table, write and read records both via the `mnesia_session` and the `mnesia_corba_session` interfaces.

Data Definition

A simple database has been created, with people as the subject. The person record containing other data types will be demonstrated in the rest of the examples. To begin with we define the data types that will be used in IDL.

```
//
// A simple example on different structures used to access mnesia from
// the outside
//
// If the tables and structures should be accessed through a corba
// interface, they need to be defined in a IDL specification so that
// the mnesia_session interface can access the types needed to use
// corba type any.
```

```
//

module persons
{
  enum Sex {male, female};

  struct data
  {
    Sex sex;
    long age;
    long phone;
  };

  struct person
  {
    string name;                 // Used as key
    data personData;             // A reference to other data structure
    string married_to;           // A relation
    sequence <string> children;  // A list
  };

};
```

Observe that the struct person, when mapping from IDL to Erlang, is prefixed with the module name followed by '_'. In this example, the table name would be `persons_person`. If this is not wanted the struct definition should be placed on the top level, i.e. above the module definition.

1.4.2 A Session Example in Erlang

A simplified example of a client written in Erlang to access `Mnesia` via the session interface is shown below.

Compile the person data definitions with:

```
erlc +'{be, erl_genserv}' person.idl
```

Include the necessary files:

```
%% Include to get the mnesia types, used in create table
-include_lib("mnesia_session/include/mnesia.hrl").

%% Include my own definitions
-include("persons.hrl").
```

Find the connector and create a private session:

```
start_session(Node) ->
    ConnPid = rpc:call(Node, mnesia_session_lib, lookup_connector, []),
    %% Use the factory to create a session
    SessPid = mnesia_connector:connect(ConnPid),
    {ConnPid, SessPid}.
```

In this example, we used `mnesia_session_lib:lookup_connector/0` but it is also possible to use `erlang:whereis(mnesia_connector)`. In Erlang, objects are represented as processes. To access the method of the object, call the method where the first argument is the object reference, as in `mnesia_connector:connect(ConnPid)`.

All functions in the IDL specification have an additional first argument which is the object reference. More information is given in the `IC` and `Orber` documentation.

To create a table with the structure defined in the IDL definition, we use the function create_person_table.

Define the table properties and call the function:

```
create_person_table(ObjKey) ->
    %% Define the table properties
    Attrs = [atom_to_list(F) || F <- record_info(fields, persons_person)],
    TabDef = #mnesia_TableDef{type = bag, mode = read_write,
                              ram_copies = [],
                              disc_copies = [],
                              disc_only_copies = [],
                              index_list = [4],  %% Index on married_to
                              attributes = Attrs,
                              %% OBSERVE that the record_name must be
                              %% exactly the same as the name of the
                              %% structure/record
                              record_name = "persons_person"},

    Res = mnesia_session:create_table(ObjKey, "persons", TabDef),
    case Res of
        {ok, ""} ->
            ok;
        Else->
            io:format("ERROR: ~s~n", [Else])
    end.
```

The example insert and a read operation looks like:

```
insert_person(SessionKey, Name, Sex, Age, Phone, Mt, Ch)
  when list(Name), atom(Sex), integer(Age),
       integer(Phone), list(Mt), list(Ch) ->

    Data = #persons_data{sex = Sex, age = Age, phone = Phone},
    Person = #persons_person{name = Name, personData = Data,
                             married_to = Mt, children = Ch},

    {ok, ""} = mnesia_session:dirty_write(SessionKey, "persons", Person),
    Person.

get_person(SessionKey, Name) when list(Name) ->
    {ok, RObj, ""} =
        mnesia_session:dirty_read(SessionKey, "persons", Name),
    hd(RObj).
```

1.4.3 A Session Example in Java

Compile the person data definitions:

```
erlc +'{be, java}' person.idl
```

Import the necessary files.

```
import com.ericsson.otp.ic.*;
```

Find the connector and create a private session:

```
  public static mnesia._sessionStub start_session(String args[])
    {
      String SNode = new String(args[0]);
      String PNode = new String(args[1]);
      String Cookie = new String(args[2]);

      mnesia._connectorStub mccRef = null;
      mnesia._sessionStub mcsRef = null;

      try
        {

          mccRef = new mnesia._connectorStub(SNode+"0",PNode,Cookie,
                                             "mnesia_connector");

          com.ericsson.otp.ic.Pid mccPid = mccRef.connect();

          mcsRef = new mnesia._sessionStub(SNode+"1",PNode,Cookie,mccPid);

          return mcsRef;
        }
      catch(Exception se)
        {
          System.out.println("Unexpected exception: " + se.toString());
          se.printStackTrace();
          return null;
        }
    }
```

To create a table with the structure defined in the IDL definition, we use the function create_person_table. Define the table properties and call the function.

```
  public static void create_person_table(mnesia._sessionStub mcsRef)
    {
      try {

        String name = "persons";
        mnesia.TableDef def = new mnesia.TableDef();
        def.type = mnesia.SetOrBag.bag;
        def.mode = mnesia.AccessMode.read_write;
```

```
      def.ram_copies = new String[0];
      def.disc_copies = new String[0];
      def.disc_only_copies = new String[0];
      int[] idxs = new int[1];
      idxs[0] = 4;
      def.index_list = idxs;
      String[] attrs = new String[4];
      attrs[0] = "name";
      attrs[1] = "personData";
      attrs[2] = "married_to";
      attrs[3] = "children";
      def.attributes = attrs;
      def.record_name = "persons_person"; // The used IDL type

      StringHolder reason;
      reason = new StringHolder();

      if(mnesia.Status.ok != mcsRef.create_table(name, def, reason))
        System.out.println("Create Table Error " + reason.value);
    }
    catch(Exception se)
      {
        System.out.println("Unexpected exception: " + se.toString());
        return;
      }
  }
```

The example insert and a read operation looks like:

```
  public static void insert_person(mnesia._sessionStub mcsRef,
                                   String name,
                                   persons.Sex sex,
                                   int age,
                                   int phone,
                                   String mt,
                                   java.lang.String[] children)
    {
      persons.data data;
      data = new persons.data(sex, age, phone);
      persons.person person = new persons.person();
      person.name = name;
      person.personData = data;
      person.married_to = mt;
      person.children = children;

      try
        {
          StringHolder reason = new StringHolder();
          Term object = new Term();
          persons.personHelper.insert(object,person);

          if(mnesia.Status.ok != mcsRef.dirty_write("persons", object, reason))
            System.out.println("Insert person Error " + reason.value);
```

```
      }
    catch(Exception se)
      {
        System.out.println("Unexpected exception: " + se.toString());
        return;
      }
  }

public static persons.person
  get_person(mnesia._sessionStub mcsRef, String name)
  {
    try
      {
        StringHolder reason = new StringHolder();
        Term key = new Term();
        mnesia.RecordlistHolder res = new mnesia.RecordlistHolder();
        key.insert_string(name);

        if(mnesia.Status.ok == mcsRef.dirty_read("persons",
                                                 key, res, reason))
          {
            if(res.value.length > 0)
              {
                persons.person rec1 =
                  persons.personHelper.extract(res.value[0]);

                return rec1;
              }
            else
              return null;
          }
        else
          {
            System.out.println("Insert person Error " + reason.value);
            return null;
          }
      }
      catch(Exception se)
        {
          System.out.println("Unexpected exception: " + se.toString());
          return null;
        }
  }
```

Note: the usage of `ETERM` differs a little to the CORBA case, when using Java it mapps to the `Term` class which inherits from `Any` class.

1.4.4 A Session Example in C

Compile the person data definitions:

```
erlc +'{be, c_genserv}' person.idl
```

Include the necessary files.

```
#include <erl_interface.h>
#include "mnesia_session.h"
#include "mnesia_connector.h"
#include "persons.h"
#include <ic.h>
```

Find the connector and create a private session:

```
erlang_pid start_session(CORBA_Environment * env)
{
    erlang_pid session_pid;

    session_pid = mnesia_connector_connect(NULL, env);

    if(env->_major != CORBA_NO_EXCEPTION)
        handle_error("connector_connect", env);
    return session_pid;
}
```

To create a table with the structure defined in the IDL definition, we use the function create_person_table. Define the table properties and call the function.

```
void create_person_table(CORBA_Environment * env)
{
    int err;
    char * reason;
    mnesia_Status result;
    mnesia_TableDef tabdef;
    mnesia_Indices idxs;

    long idx_list[1] = {4};
    char * attrs[4] = {"name", "personData", "married_to", "children"};

    tabdef.type = mnesia_bag;
    tabdef.mode = mnesia_read_write;

    tabdef.ram_copies._maximum = 0;
    tabdef.ram_copies._length = 0;
    tabdef.ram_copies._buffer = NULL;

    tabdef.disc_copies._maximum = 0;
    tabdef.disc_copies._length = 0;
    tabdef.disc_copies._buffer = NULL;

    tabdef.disc_only_copies._maximum = 0;
    tabdef.disc_only_copies._length = 0;
    tabdef.disc_only_copies._buffer = NULL;

    tabdef.index_list._maximum = 5;
    tabdef.index_list._length = 1;
```

```
    tabdef.index_list._buffer = idx_list;

    tabdef.attributes._maximum = 5;
    tabdef.attributes._length = 4;
    tabdef.attributes._buffer = attrs;

    tabdef.record_name = "persons_person"; /* The name of the stored
                                              type/struct */

    result = mnesia_session_create_table(NULL, "persons", &tabdef, &reason, env);

    if(env->_major != CORBA_NO_EXCEPTION)
    {
        fprintf(stderr,"
 error in create_table: %d
", env->_major);
        exit(1);
    }
    else if(result != mnesia_ok)
    {
        fprintf(stderr, "Create table failed with reason %s", reason);
    }
    CORBA_free(reason);
}
```

The example insert and a read operation looks like:

```
void insert_person(CORBA_Environment * env,
                   char *name, persons_Sex sex, int age,
                   int phone, char * mt, persons_person_children * ch)
{
    ETERM *person, *children, *temp;
    int err, i;
    char * reason;
    mnesia_Status result;
    extern char * oe_persons_Sex[];

    children = erl_mk_empty_list();

    if(ch != NULL)
        for(i = 0; i < ch->_length; i++)
        {
            temp = erl_mk_string(ch->_buffer[i]);
            children = erl_cons(temp, children);
            erl_free_term(temp);
        };

    person = erl_format("{persons_person,~s,{persons_data,~a,~i,~i}, ~s, ~w}",
                        name, oe_persons_Sex[sex], age, phone, mt, children);

    result = mnesia_session_dirty_write(NULL, "persons", person, &reason, env);

    if(env->_major != CORBA_NO_EXCEPTION)
```

```
    {
        fprintf(stderr,"
 error in insert_person: %d
", env->_major);
        exit(1);
    }
    else if(result != mnesia_ok)
    {
        fprintf(stderr, "Insert person failed with reason %s", reason);
        exit(-1);
    }

    erl_free_term(children);
    erl_free_term(person);

    CORBA_free(reason);
}

void get_person(CORBA_Environment * env, char * name, persons_person * person)
{
    int err, i;
    char * reason;
    mnesia_Status result;
    mnesia_Recordlist *rec_list;
    ETERM * key;

    key = erl_mk_string(name);

    result = mnesia_session_dirty_read(NULL, "persons", key, &rec_list,
                                       &reason, env);
    if(env->_major != CORBA_NO_EXCEPTION)
    {
        fprintf(stderr,"
 error in get_person: %d
", env->_major);
        exit(1);
    }
    else if(result != mnesia_ok)
    {
        fprintf(stderr, "Get person failed with reason %s", reason);
        exit(-1);
    }

    if(rec_list->_length > 0)
    {
        /* Only interrested in the first match */
        decode_person(rec_list->_buffer[0], person);
        for(i=0; i < rec_list->_length; i++)
            erl_free_term(rec_list->_buffer[i]);
    }
    else
    {
        fprintf(stderr, "Insert person failed empty_list returned ");
```

```
        exit(-1);
    }

    CORBA_free(rec_list);
    CORBA_free(reason);
    erl_free_term(key);
}
```

Note: the usage of ETERM differs when using C as the user must develop their own code to create and extract information from the ETERM structures, whereas, CORBA does this automatically.

1.5 CORBA Session Examples

1.5.1 A CORBA Session in Erlang

Following is a simplified example of a client written in Erlang to access Mnesia via the CORBA session interface. The type definitions are the same as in the previous chapter [page 13].

To begin with include the necessary files.

```
%% Include to get the mnesia types, used in create table
-include_lib("mnesia_session/include/mnesia.hrl").

%% Include to get the corba types i.e. any
-include_lib("orber/include/corba.hrl").

%% Include my own types
-include("persons.hrl").
```

Find the connector and create a private session.

```
start_corba_session(Host, Port) ->
    %% Lookup the inital corba name sever
    Addr = "iiop://" ++ atom_to_list(Host) ++ ":" ++ integer_to_list(Port),
    NS = corba:resolve_initial_references_remote("NameService", [Addr]),

    %% Create a corba name object
    NC = lname_component:set_id(lname_component:create(),
                                "mnesia_corba_connector"),
    N = lname:insert_component(lname:create(), 1, NC),

    %% Lookup the object reference to the factory mnesia_corba_connector
    Cok = 'CosNaming_NamingContext':resolve(NS, N),

    %% Use the factory to create a session
    Sok = mnesia_corba_connector:connect(Cok),
    {Cok, Sok}.
```

In Erlang, objects are represented as processes. To access the method of the object, call the method where the first argument is the object reference, as in `mnesia_connector:connect(ConnPid)`. All functions in the IDL specification have an additional first argument which is the object reference, more information is given in the `IC` and `Orber` documentation.

To create a table with the structure defined in the IDL definition, we use the function create_person_table.

Define the table properties and call the function:

```
create_person_table(ObjKey) ->

    %% Define the table properties
    Attrs = [atom_to_list(F) || F <- record_info(fields, persons_person)],
    TabDef = #mnesia_TableDef{type = bag, mode = read_write,
                              ram_copies = [],
                              disc_copies = [],
                              disc_only_copies = [],
                              index_list = [4],  %% Index on married_to
                              attributes = Attrs,
                              %% NOTE that the record name must be exactly
                              %% the same as the name of the
                              %% structure/record to be used.
                              record_name = "persons_person"},

    Res = mnesia_corba_session:create_table(ObjKey, "persons", TabDef),
    case Res of
        {ok, ""} ->
            ok;
        Else ->
            io:format("ERROR: ~s~n", [Else])
    end.
```

In this example the insert and a read operation looks like:

```
insert_person(SessionKey, Name, Sex, Age, Phone, Mt, Ch) ->
    Data = #persons_data{sex = Sex, age = Age, phone = Phone},
    Person = #persons_person{name = Name, personData = Data,
                             married_to = Mt, children = Ch},

    Any = #any{typecode = persons_person:tc(), value = Person},
    {ok, ""} = mnesia_corba_session:dirty_write(SessionKey, "persons", Any),
    Person.

get_person(SessionKey, Name) ->
    Obj = #any{typecode = {tk_string, 0}, value = Name},
    {ok, RObjs, ""} =
        mnesia_corba_session:dirty_read(SessionKey, "persons", Obj),
    RObj = hd(RObjs),
    RObj#any.value.
```

Notice the usage of the any type in both the read and write operations. `erlang::term` in the IDL specification is represented as the CORBA type any when using the CORBA session interface. The any type is represented as a 2-tuple containing type-code and value, the user must in Erlang insert the type-code.

1.5.2 A CORBA Session in Java

This example is the same but using Java and OrbixWeb.

For this example to work, the following specifications need to be compiled:

- compilation of the user defined IDL specifications
- ${OTP_ROOT}/lib/mnesia_session-Version/src/mnesia_corba_session.idl
- ${OTP_ROOT}/lib/ic-Version/include/erlang.idl
- ${OTP_ROOT}/lib/orber-Version/COSS/CosNaming/cos_naming.idl
- ${OTP_ROOT}/lib/orber-Version/examples/Stack/InitialReferences.idl

Import CORBA and CosNaming Classes:

```
import CosNaming._NamingContextRef;
import CosNaming.Name;
import IE.Iona.Orbix2._CORBA;
import IE.Iona.Orbix2.CORBA.*;
```

Find the connector and create a private session.

```
    public static mnesia._corba_sessionRef
        start_corba_session(String args[])
    {
        mnesia._corba_sessionRef mcsRef = null;
        mnesia._corba_connectorRef mccRef = null;
        CORBA._InitialReferencesRef init;
        _NamingContextRef nsContext;
        Name name;
        _ObjectRef initRef, nsRef, objRef;
        Orber.InitialReference ir = new Orber.InitialReference();

        String srvHost = new String(args[0]);
        Integer srvPort = new Integer(args[1]);
        try
        {
            // For an explanation about initial reference handling see
            // the "Interoperable Naming Service" specification.

            // Create Initial reference (objectkey "INIT").
            String s = ir.stringified_ior(srvHost, srvPort.intValue());
            initRef = _CORBA.Orbix.string_to_object(s);
            init = CORBA.InitialReferences._narrow(initRef);
            // Fetch name service reference.
            nsRef = init.get("NameService");
            nsContext = CosNaming.NamingContext._narrow(nsRef);
            // Create a name
            name = new Name(1);
            name.buffer[0] =
                new CosNaming.NameComponent("mnesia_corba_connector", "");
            try
            {
                objRef = nsContext.resolve(name);
```

```
            }
            catch(UserException n)
            {
                System.out.println("Unexpected exception: " + n.toString());
                return null;
            }
            mccRef = mnesia.corba_connector._narrow(objRef);

            // Create and return the session reference
            mcsRef = mccRef.connect();
            return mcsRef;
        }
        catch(SystemException se)
        {
            System.out.println("Unexpected exception: " + se.toString());
            se.printStackTrace();
            return null;
        }
    }
```

Create the person table

```
    public static void create_person_table(mnesia._corba_sessionRef mcsRef)
    {
        String name = "persons";
        mnesia.TableDef def = new mnesia.TableDef();
        def.type = mnesia.SetOrBag.bag;
        def.mode = mnesia.AccessMode.read_write;
        def.ram_copies = new mnesia.NodeList(0);
        def.disc_copies = new mnesia.NodeList(0);
        def.disc_only_copies = new mnesia.NodeList(0);
        mnesia.Indices idxs = new mnesia.Indices(1);
        idxs.buffer[0] = 4;
        def.index_list = idxs;
        mnesia.AttrNames attrs = new mnesia.AttrNames(4);
        attrs.buffer[0] = "name";
        attrs.buffer[1] = "personData";
        attrs.buffer[2] = "married_to";
        attrs.buffer[3] = "children";
        def.attributes = attrs;
        def.record_name = "persons_person"; // The used IDL type

        StringHolder reason;
        reason = new StringHolder();
        try
        {
            if(mnesia.Status.ok != mcsRef.create_table(name, def, reason))
                System.out.println("Create Table Error " + reason.value);
        }
        catch( SystemException se)
        {
            System.out.println("Unexpected exception: " + se.toString());
            return;
```

```
        }
    }
```

The insert operation. Observe the encapsulation of person in the `mnesia.Record`. Java or OrbixWeb will handle the type encoding.

```
    public static void insert_person(mnesia._corba_sessionRef mcsRef,
                                     String name,
                                     int sex,
                                     int age,
                                     int phone,
                                     String mt,
                                     _sequence_String children)
    {
        persons.data data;
        data = new persons.data(sex, age, phone);
        persons.person person = new persons.person();
        person.name = name;
        person.personData = data;
        person.married_to = mt;
        person.children = children;

        try
        {
            StringHolder reason = new StringHolder();
            mnesia.Record object = new mnesia.Record();
            object.insert(person);

            if(mnesia.Status.ok != mcsRef.dirty_write("persons", object, reason))
                System.out.println("Insert person Error " + reason.value);
        }
        catch(SystemException se)
        {
            System.out.println("Unexpected exception: " + se.toString());
            return;
        }
    }
```

A read operation. Observe the extraction of person from the received `mnesia.Recordlist` and the `key` object handling.

```
    public static persons.person
        get_person(mnesia._corba_sessionRef mcsRef, String name)
    {
        try
        {
            StringHolder reason = new StringHolder();
            mnesia.Key key = new mnesia.Key();
            mnesia.Recordlist res = new mnesia.Recordlist();
            key.insertString(name);

            if(mnesia.Status.ok == mcsRef.dirty_read("persons",
```

```
                                                key, res, reason))
            {
                if(res.length > 0)
                {
                    persons.person rec1 = new persons.person();
                    res.buffer[0].extract(rec1);
                    return rec1;
                }
                else
                    return null;
            }
            else
            {
                System.out.println("Insert person Error " + reason.value);
                return null;
            }
        }
        catch(SystemException se)
        {
            System.out.println("Unexpected exception: " + se.toString());
            return null;
        }
    }
```

1.6 Mnesia Session Release Notes

This document describes the changes made to the Mnesia Session application. The intention of this document is to list all incompatibilities as well as all enhancements and bug-fixes for each and every release of Mnesia Session. Each release of Mnesia Session constitutes one section in this document. The title of each section is the version number of Mnesia Session.

1.6.1 Mnesia Session 1.1.5

Improvements and new features

The server side of Corba sessions has been made more lightweight (obtained by usage of pseudo objects in Orber).

Fixed Bugs and malfunctions

None.

Incompatibilities

None.

Known bugs and problems

None.

1.6.2 Mnesia Session 1.1.4

Improvements and new features

None

Fixed Bugs and malfunctions

Fixed error in table_info when accessing mnesia version 3.9.4 or later.

Incompatibilities

None

Known bugs and problems

None.

1.6.3 Mnesia Session 1.1.3

This release is a minor release where a new session is tested.

Improvements and new features

A new session, direct java to mnesia session is tested. A new example and documentation is added.

Fixed Bugs and malfunctions

None

Incompatibilities

None

Known bugs and problems

None.

1.6.4 Mnesia Session 1.1.2

This release is a minor release and the release notes describes the difference between version 1.1.1 and version 1.1 of Mnesia Session.

Improvements and new features

Copyright comments have been changed.

Fixed Bugs and malfunctions

None

Incompatibilities

None

Known bugs and problems

None.

1.6.5 Mnesia Session 1.1.1

This release is a minor release and the release notes describes the difference between version 1.1.1 and version 1.1 of Mnesia Session.

Improvements and new features

- Minor documentation changes.
- Generated src files with the latest version of IC.

Fixed Bugs and malfunctions

None

Incompatibilities

None

Known bugs and problems

None.

1.6.6 Mnesia Session 1.1

This release is a minor release and the release notes describes the difference between version 1.1 and version 1.0 of Mnesia Session.

Improvements and new features

- `wait_for_tables` now has an extra out parameter, failed_tabs, which contains the tables which have not yet been loaded if the returned `status` is `timeout`, this corresponds with Mnesia's interface.
- The 'C' example in the documentation has been updated with the new (CORBA like) look of 'C' interfaces generated with IC-3.1.2.

Fixed Bugs and malfunctions

- The `dump_log` function missed an out parameter (`reason`) in the IDL specification, see incompatibilities.

Incompatibilities

- To comply with the rest of the IDL specification an out parameter has been added to the function `dump_log`.
- `wait_for_tables` has a new parameter.

Known bugs and problems

None.

1.6.7 Mnesia Session 1.0

This release is a minor release and the release notes describes the difference between version 1.0 and version 0.81 of Mnesia Session.

Improvements and new features

- The documentation has been enhanced.
- The interface has been reworked internally to comply to Mnesia 3.4, more specifically, the record and table name relation has changed, see incompatibilities below.

Fixed Bugs and malfunctions

None.

Incompatibilities

- `dirty_write, dirty_delete_object, dirty_match_object` functions have a new in parameter (Table Name). The table name parameter is needed in Mnesia Session to be able to differentiate between the table names and record names.

Known bugs and problems

No new bugs or problems. See earlier release notes.

1.6.8 Mnesia Session 0.81

This release is a minor release and the release notes describes the difference between version 0.81 and version 0.80 of Mnesia Session.

Improvements and new features

None.

Fixed Bugs and malfunctions

- Two modules were missing in `mnesia_session.app`

Incompatibilities

None.

Known bugs and problems

- The documentation is not ready.
- The interface to `Mnesia Session` may be an object for change in future releases.

1.6.9 Mnesia Session 0.8

`Mnesia` is written in Erlang and intended to be used with Erlang applications.

`Mnesia Session` is a new application which enables access to the Mnesia DBMS from foreign programming languages (i.e. other languages than Erlang). The `Mnesia Session` interface is defined in `IDL` (an Interface Definition Language standardized by `OMG` (the Object Management Group)).

Improvements and new features

- `Mnesia Session` contains two back-ends. One for access via the standardized `IIOP` protocol and the other for access via Erlang's proprietary distribution protocol.

Following is a brief summary of the available configuration parameters:

- debug ::= none | verbose | debug | trace
- enable_corba ::= false | true
- corba_connector_name ::= A valid COSS name

Fixed Bugs and malfunctions

N.A.

Incompatibilities

N.A.

Known bugs and problems

- The documentation is not ready.
- The interface to `Mnesia Session` may be an object for change in future releases.

Any comments regarding the Mnesia Session application would be appreciated.

Mnesia_Session Reference Manual

Short Summaries

- Erlang Module mnesia_session [page 34] – Mnesia_Session

mnesia_session

No functions are exported.

mnesia_session

Erlang Module

Mnesia is written in Erlang and intended to be used for Erlang applications.

The Mnesia_Session application enables access to the Mnesia DBMS from foreign programming languages (i.e. other languages than Erlang). The Mnesia_Session interface is defined in IDL (an Interface Definition Language standardized by OMG (the Object Management Group)).

This module is automatically generated from the IDL specification `mnesia_session.idl` and is documented in the IDL specification.

Configuration parameters

Mnesia_Session reads the following application configuration parameters:

- `-mnesia_session corba_connector_name Name`. This parameter controls which initial name Mnesia Corba Connector is bound. This parameter is only used when the `enable_corba` is set to true. The name is validated by using `lname:check_name(Name)`. See the `Orber` application for details. The default name is {`'CosNaming_NameComponent'`, `"IDL:CosNaming/NameComponent:1.0", "mnesia_corba_connector", []`}.
- `-mnesia_session debug Level` Controls the debug level of Mnesia_Session. Possible values are:

  `none` No trace outputs at all. This is the default.

  `verbose` Activates tracing of important debug events.

  `debug` Activates all events at the verbose level and traces of all debug events.

  `trace` Activates all debug events both verbose and debug as well as trace events that provide internal printouts.
  This level is only intended for debugging of small toy systems since many large events may be generated.
- `-mnesia_session enable_corba true | false`. This flag enables Mnesia_Session to access Mnesia via CORBA. If CORBA access is enabled, the application Orber must be started in order to start Mnesia Session. The default is `false`.

First the SASL application parameters are checked, then the command line flags are checked, and finally, the default value is chosen.

See Also

mnesia(3), orber(3), application(3)