

Contributed Libraries

Denys Duchier
Leif Kornstaedt

Version 1.2.3
December 1, 2001



Credits

Mozart logo by Christian Lindig

License Agreement

This software and its documentation are copyrighted by the German Research Center for Artificial Intelligence (DFKI), the Swedish Institute of Computer Science (SICS), and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE AND ITS DOCUMENTATION ARE PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Contents

1	Regular Expressions	1
2	GDBM Database Interface	5
2.1	Interface	5
2.2	Installation	7
3	Backwards Compatibility	9
3.1	The <code>TextPickle</code> module	9
4	An Example of a Server Directory	11
4.1	The ExampleDirectory Module	11

Regular Expressions

The `regex` module implements an interface to the POSIX regex library and builds some higher level facilities on top of it. The module may be imported as follows:

```
import Regex at 'x-oz://contrib/regex'
```

`Regex.is`

```
{Regex.is +X ?B}
```

Returns `true` iff X is a regex object.

`Regex.make`

```
{Regex.make +PAT ?RE}
```

Creates regex object RE from virtual string pattern PAT.

`Regex.search`

```
{Regex.search +RE +TXT ?MATCH}
```

Returns the next MATCH of regex RE in virtual string TXT, or `false` if there is no such match. RE is also permitted to be a virtual string pattern, in which case it is automatically compiled into a regex object.

A match is a record with integer features: one for each group in the pattern and also feature 0 for the whole match. The value on each feature is a pair `I#J` of start and end indices into TXT. If TXT is a byte string, you can simply invoke `{ByteString.slice TXT I J}` to extract the match.

`Regex.group`

```
{Regex.group +N +MATCH +TXT ?GROUP}
```

Return the substring GROUP matched by the N group of MATCH in virtual string TXT. MATCH should be the result of calling `Regex.search` on TXT.

`Regex.groups`

```
{Regex.groups +MATCH +TXT ?GROUPS}
```

Returns the list GROUPS of substrings of TXT corresponding to the groups of MATCH. Group 0 (the whole match) is not included.

`Regex.allMatches`

```
{Regex.allMatches +RE +TXT ?MATCHES}
```

Returns the list of all MATCHES of regular expression RE in virtual string TXT. RE should not be anchored.

`Regex.forAll`

```
{Regex.forAll +RE +TXT +P}
```

Applies the 1 argument procedure P to every match of RE in TXT.

`Regex.map`

```
{Regex.map +RE +TXT +F ?RESULTS}
```

Applies the 1 argument function F to every match of RE in TXT and returns the corresponding list of RESULTS.

`Regex.foldR`

```
{Regex.foldR +RE +TXT +F INIT ?RESULT}
```

`Regex.foldL`

```
{Regex.foldL +RE +TXT +F INIT ?RESULT}
```

The usual reduction procedure (see List module).

`Regex.split`

```
{Regex.split +RE +TXT ?STRINGS}
```

Splits the input TXT at every match of separator RE, and returns the resulting list of strings.

`Regex.compile`

```
{Regex.compile +PAT +CFLAGS ?RE}
```

This is the more complicated version of `Regex.make`. The additional CFLAGS argument further parametrizes the regex compilation process. It is either an atom or a list of atoms, from the set: `extended`, `icase`, `nosub`, `newline`. The default is `[extended newline]`. See the man page for `regcomp` for further details.

`Regex.execute`

```
{Regex.execute +RE +TXT +IDX +EFLAGS ?MATCH}
```

This is the more complicated version of `Regex.search`. Integer IDX is the offset at which to start the search in +TXT. EFLAGS further specify how to search: it is an atom or list of atoms, from the set: `notbol`, `noteol`. The default is `nil`. See the man page for `regexexec` for further details.

`Regex.cflags.set`

`Regex.cflags.get`

```
{Regex.cflags.set +SPEC}
{Regex.cflags.get ?SPEC}
```

Set or get the current CFLAGS defaults, e.g. used by `Regex.make`.

`Regex.eflags.set`

`Regex.eflags.get`

```
{Regex.eflags.set +SPEC}
{Regex.eflags.get ?SPEC}
```

Set or get the current EFLAGS defaults, e.g. used by `Regex.search`.

Regex.replace

```
{Regex.replace +TXT +RE +FUN ?RES}
```

Replace every occurrence of RE in TXT with the result of applying FUN to the current TXT and the current match.

Regex.replaceRegion

```
{Regex.replaceRegion +TXT +LO +HI +RE +FUN ?RES}
```

Same as above, but only operate on the region starting at LO inclusive and ending at HI exclusive.

GDBM Database Interface

The `gdbm` module implements an interface to the GNU GDBM database library and builds some higher level facilities on top of it.

2.1 Interface

The module may be imported as follows:

```
import Gdbm at 'x-oz://contrib/gdbm'
```

The interface provided is similar to dictionaries and allows to store stateless Oz terms under keys. A key is an arbitrary virtual string.

`Gdbm.is`

```
{Gdbm.is +X ?B}
```

Returns `true` iff X is a gdbm object.

`Gdbm.new`

```
{Gdbm.new +R ?DB}
```

This is the convenient way of creating a gdbm object DB. R is a record that describes how to open the database. The label is the opening method, e.g. `read`, `write`, `create`, or `new`. The first argument of the record is the file name. For example `read('/usr/local/people.db')` asks to open the database located in `/usr/local/people.db` for reading only. `create('~data')` opens or creates the database `data` in the user's home directory and opens it both for reading and writing; `new('~data')` is similar, but overwrites it if it already existed.

Optional feature `mode` indicates with what permissions the file should be created (this of course is only relevant for creating a new database). The mode may be specified as an integer, with the usual Unix meaning. It may also be specified symbolically, as a record or list of records:

```
[owner group(read)]
```

This indicates that the owner has all permission rights and that group members are granted read access. Others have no rights. To also grant write access to group members, you could say:

```
[owner group(read write)]
```

The default is 0644: owner has read and write access; all others have read access.

Optional feature boolean `fast` requests updates without disk synchronization. See the GDBM documentation for details. Default is `false`.

`Gdbm.open`

```
{Gdbm.open +FILE +FLAGS +MODE +BLOCK ?DB}
```

This is the more complicated way of opening a gdbm database. The `FLAGS` specify the opening method. This is an atom or list of atoms from the set: `read`, `write`, `create` (equivalently `new` or `truncate`). It may also include the atom `fast` (see above). Each symbol may be abbreviated to its initial letter. `MODE` was described above. `BLOCK` is an integer for the block size of transfers (see GDBM documentation): use 0 to get a system dependent appropriate default.

`Gdbm.get`

```
{Gdbm.get +DB +KEY ?VAL}
```

Retrieves the Oz value `VAL` stored under `KEY`. The latter may be an arbitrary virtual string. If there is no such `KEY` in `DB`, an exception is raised.

`Gdbm.condGet`

```
{Gdbm.condGet +DB +KEY DEFAULT ?VAL}
```

Similar to the above, but returns `DEFAULT` if there is no such `KEY` in `DB`.

`Gdbm.put`

```
{Gdbm.put +DB +KEY +VAL}
```

Stores Oz value `VAL` under `KEY` in `DB`. `VAL` must be ground and stateless.

`Gdbm.firstkey`

```
{Gdbm.firstkey +DB ?KEY}
```

Returns a `KEY` in `DB` (see below). There is absolutely no guarantee as to which key this is going to be. Returns `unit` if the database is empty.

`Gdbm.nextkey`

```
{Gdbm.nextkey +DB +KEY ?NEXT}
```

Returns the `NEXT` key after `KEY`. Again, there is no guarantee as to which key this is going to be. The only guarantee is that if you continue in this manner, you will enumerate all the keys in the database, in some order, without repeats. Returns `unit` when there are no more keys.

`Gdbm.close`

```
{Gdbm.close +DB}
```

Closes the database. Subsequent access attempts raise an exception.

`Gdbm.remove`

```
{Gdbm.remove +DB +KEY}
```

Deletes the entry for `KEY` if it exists.

`Gdbm.reorganize`

```
{Gdbm.reorganize +DB}
```

see the GDBM documentation.

Gdbm.keys

```
{Gdbm.key +DB ?KEYS}
```

Returns the lazy list of all KEYS in the database.

Gdbm.entries

```
{Gdbm.entries +DB ?ENTRIES}
```

Returns the lazy list of all pairs KEY#VALUE in the database.

Gdbm.items

```
{Gdbm.items +DB ?ITEMS}
```

Returns the lazy list of all values stored in the database.

Gdbm.forAll

```
{Gdbm.forAll +DB +P}
```

Calls {P VALUE} for every entry in the database.

Gdbm.forAllInd

```
{Gdbm.forAllInd +DB +P}
```

Calls {P KEY VALUE} for every entry in the database.

2.2 Installation

The GDBM library must be available on your system as a shared object library. For Linux systems, this is normally the case. For other systems, it may need to be built and installed. Newer versions of GDBM (e.g. 1.8.0) will automatically build a shared object library if that is possible on your system. Thus you should download the most recent version of GDBM from <ftp://ftp.gnu.org/gnu/gdbm/>¹ and install it on your system.

If, for some bizarre reason, you must use an older version, the standard distribution needs to be patched to compile for *position independent code* and to create a shared object library. Thus, for the older version 1.7.3, we provide a fix for the standard gdbm distribution: its purpose is to automate the creation of the shared object library. Both the standard gdbm 1.7.3 distribution and our fix can be downloaded from our ftp server:

- <ftp://ftp.mozart-oz.org/pub/mozart/extras/gdbm-1.7.3.tar.gz>²
- <ftp://ftp.mozart-oz.org/pub/mozart/extras/gdbm-1.7.3-fix.tar.gz>³

They both unpack into the same directory (namely `gdbm-1.7.3`). You should unpack the standard distribution first, then our fix since it overwrites certain files.

¹<ftp://ftp.gnu.org/gnu/gdbm/>

²<ftp://ftp.mozart-oz.org/pub/mozart/extras/gdbm-1.7.3.tar.gz>

³<ftp://ftp.mozart-oz.org/pub/mozart/extras/gdbm-1.7.3-fix.tar.gz>

```
tar zxf gdbm-1.7.3.tar.gz
tar zxf gdbm-1.7.3-fix.tar.gz
cd gdbm-1.7.3
./configure
make
make install
```

We also make available the following targets: `install.so`, `install.h`, `install.man`, `install.info`.

If you install in a non-standard directory, you may have to set `LD_LIBRARY_PATH` appropriately so that `libgdbm.so` may be found.

Backwards Compatibility

The modules described in this chapter provide functionality to increase interoperability with or upgrade from previous releases of Mozart.

3.1 The `TextPickle` module

The `TextPickle` module allows one to read text pickles written by Mozart 1.0.1. It may be imported as follows:

```
import TextPickle at 'x-oz://contrib/compat/TextPickle.ozf'
```

`TextPickle.load`

```
{TextPickle.load +V X ?YTs}
```

takes the name of a file `V` containing a text pickle and parses it. Returns the value represented by the pickle in `X`. `X` may have holes (unbound variables) in place of the data structures that could not be converted (at the moment, these include procedures and extensions except byte strings). `YTs` maps these holes to low-level descriptions of the corresponding data structures as described by the pickle; for procedures, this includes the instructions making up the body.

An Example of a Server Directory

This contribution is meant primarily to work as an example of how one can use the module `Discovery` and build a yellow pages service. The directory service consists out of three parts: a directory server, an interface to announce services, and a way to find announced services.

All services have descriptions. When a lookup is performed, a pattern is given to specify what to look for. The result of the lookup are services that have descriptions matching the pattern. Both descriptions and patterns are Oz records.

So what does it mean that a description match a pattern? All features in a pattern must be present in the description. If a field in the pattern has a value, then it must be either equal or match to a corresponding field in the description. For example, `f(a:1 b:g(x:5 y:7) c:28 d:foo)` does match `f(c:28 b:g d:_)`, but not the other way around.

One could say that a pattern puts constraints on a matching description. Our form of pattern puts constraints on the label and features of the description.

4.1 The ExampleDirectory Module

The module has three classes under features (`server`, `serviceProvider` and `client`), which corresponds to the three parts mentioned earlier.

The class `ExampleDirectory.server` has following methods:

```
init
    init(port:PortNr <= useDefault
        expectedClientId:ClientId <= unit
        id:ServerId <= directory(kind:recordMatching))
```

This method starts a directory server. The initial contact between a client (or service provider) and the server is made through the ip port `PortNr`. The default ip port number is `Discovery.defaultPortNumber`. After the first contact has been performed, all subsequent interactions are done using Oz ports.

`ServerId` is a record explaining which type of directory server it is.

`ClientId` is a record used to specify which clients should be served. If a client id sent from a client (or service provider) matches `ClientId`, the client will be served. If the feature `expectedClientId` is not specified the server will serve every client requesting service.

close

```
close()
```

Closes the operation of the server.

The class `ExampleDirectory.serviceProvider` has following methods:

init

```
init(serverPort:PortNr <= useDefault
      id:ClientId <= unit
      expectedServerId:ServerId <= directory(kind:recordMatching)
      timeOut:TimeOut <= 1000)
```

This method broadcasts to the ip port `PortNr` and waits for answers from servers that listen to it. The servers that answer and which id matches `ServerId` will be used by the other methods of this class.

When interacting with the servers, `ClientId` will be used as identification. The method will not consider answers received after `TimeOut` milliseconds has gone by.

add

```
add(description:Desc
      ticket:Ticket
      key:?Key <= _)
```

This method announces a service with a description `Desc`. `Ticket` is a ticket to an Oz-entity that is used as an interface to the service. An Oz name that can be used to remove the service from the directories is returned.

remove

```
remove(key:Key)
```

Tells the servers to remove a service with the key `Key`.

The class `ExampleDirectory.client` has following methods:

init

```
init(serverPort:PortNr <= useDefault
      id:ClientId <= unit
      expectedServerId:ServerId <= directory(kind:recordMatching)
      timeOut:TimeOut <= 1000)
```

Similar to `ExampleDirectory.serviceProvider`.

lookup

```
lookup(pattern:Pattern services:?Services)
```

Asks servers for services that match the `Pattern`. A list of pairs is returned. Those pairs consist of a service description and its ticket.