Entwicklung eines dezentralen aktiven Dokumentenspeichersystems für digitale Bibliotheken

Diplomarbeit

(Band 2 - Quellcode der Implementierung)

eingereicht bei

Prof. Dr. O. Drobnik
Professur für Telematik und verteilte Systeme
Johann Wolfgang Goethe - Universität
Frankfurt am Main

von

Hans Matzen
Henry-Budge-Str. 58
60320 Frankfurt/Main

Inhaltsverzeichnis

Anhang D

Quellcode des Pythonmoduls zur Anbindung der PostgreSQL Datenbank

Zur Installation des Python-Moduls zur PostgreSQL-Datenbankanbindung, muß sowohl Python als auch PostgreSQL installiert sein. Innerhalb des Python Sourcetrees, muß in die Datei Modules/Setup.local folgende Zeile eingetragen werden:

```
pgpy pgpylib.o -I<PostgreSQL-Include-Pfad> \
              -L<PostgreSQL-Library-Pfad> -lpq
```

Danach einfach make, der neue Pythoninterpreter erlaubt es dann mittels: `import pgpy` auf das Modul zuzugreifen. Die Methoden der drei Objekte (pgconn, pgres, pgblob) sind den von der PostgreSQL C-Library bereitgestellten Funktionen nachempfunden. Bei Unsicherheiten können Informationen zur jeweiligen Methode über die `.__doc__` Eigenschaft der Methoden abgerufen werden. Beispiele für den Einsatz finden sich in ausreichender Anzahl in Anhang E.

D.1 pgpylib.h

```
/* Python Postgres Interface Library
   (C) by Hans Matzen, 1997, Frankfurt, Germany

*/

/* this is pgpylib.h */
```

```
#include <stdlib.h>
#include "Python.h"
#include "libpq-fe.h"

static PyObject *ErrorObject;

#define Py_Try(BOOLEAN) { if (!(BOOLEAN)) return NULL;}


/* ------------------------------------------------------------- */
/* everything for pgresult, the postgres result handle object     */
/* ------------------------------------------------------------- */

/* the definitions for the python object */
typedef struct {
  PyObject_HEAD
  PGresult *res;
} pgres;

staticforward PyTypeObject pgres_obj;

/* all that help stuff */


static char pgres_obj__doc__[]=
   "Postgres Result Handle Object";

static char pgpy_pgres__doc__[]=
   "pgres(res)=pgres";

static char pgres_get_tuple_count__doc__[]=
   "returns the number of tuples.";
static char pgres_get_field_count__doc__[]=
   "returns the number of fields.";
static char pgres_get_fieldname_byindex__doc__[]=
   "returns the fieldname.";
static char pgres_get_fieldindex_byname__doc__[]=
   "returns the fieldname.";
static char pgres_get_fieldtype__doc__[]=
   "returns the fieldtype.";
static char pgres_get_fieldsize__doc__[]=
   "returns the fieldsize.";
static char pgres_get_fieldvalue__doc__[]=
```

```
   "returns the fieldvalue.";
static char pgres_fieldlength__doc__[]=
   "returns the fieldlength.";
static char pgres_commandstatus__doc__[]=
   "returns the commandstatus.";
static char pgres_objectstatus__doc__[]=
   "returns the objectstatus.";
static char pgres_get_resultlist__doc__[]=
   "prints the query result.";

/* forward declarations for methods */

staticforward pgres   * new_pgres(pgres *init);
staticforward void      free_pgres(pgres *self);
staticforward
   PyObject* pgpy_pgres(PyObject *self,PyObject *args);
staticforward
   PyObject* pgres_getattr(pgres *self,char *name);
staticforward PyObject* pgres_get_tuple_count(pgres *self);
staticforward PyObject* pgres_get_field_count(pgres *self);
staticforward
   PyObject* pgres_get_fieldname_byindex(pgres *self,PyObject *args);
staticforward
   PyObject* pgres_get_fieldindex_byname(pgres *self, PyObject *args);
staticforward
   PyObject* pgres_get_fieldtype(pgres *self, PyObject *args);
staticforward
   PyObject* pgres_get_fieldsize(pgres *self,PyObject *args);
staticforward
   PyObject* pgres_get_fieldvalue(pgres *self,PyObject *args);
staticforward
   PyObject* pgres_get_fieldlength(pgres *self, PyObject * args);
staticforward PyObject* pgres_commandstatus(pgres *self);
staticforward PyObject* pgres_objectstatus(pgres *self);
staticforward
   PyObject* pgres_get_resultlist(pgres *self,PyObject *args);


/* method table for class pgres */

static struct PyMethodDef pgres_methods[]={
  {"get_tuple_count"      ,(PyCFunction)pgres_get_tuple_count       },
  {"get_field_count"      ,(PyCFunction)pgres_get_field_count       },
  {"get_fieldname_byindex",(PyCFunction)pgres_get_fieldname_byindex },
  {"get_fieldindex_byname",(PyCFunction)pgres_get_fieldindex_byname },
```

```
  {"get_fieldtype"        ,(PyCFunction)pgres_get_fieldtype         },
  {"get_fieldsize"        ,(PyCFunction)pgres_get_fieldsize         },
  {"get_fieldvalue"       ,(PyCFunction)pgres_get_fieldvalue        },
  {"get_fieldlength"      ,(PyCFunction)pgres_get_fieldlength       },
  {"commandstatus"        ,(PyCFunction)pgres_commandstatus         },
  {"objectstatus"         ,(PyCFunction)pgres_objectstatus          },
  {"get_resultlist"       ,(PyCFunction)pgres_get_resultlist        },
  {NULL,NULL}
};


/* pgres object table  */

static PyTypeObject pgres_obj= {
  PyObject_HEAD_INIT(&PyType_Type) 0,
  "pgres",
  sizeof(pgres),
  0,
  (destructor)free_pgres,
  (printfunc)0,
  (getattrfunc)pgres_getattr,
  (setattrfunc)0,
  (cmpfunc)0,
  (reprfunc)0,
  0,0,0,
  (hashfunc)0,
  (ternaryfunc)0,
  (reprfunc)0,
  0L,0L,0L,0L,
  pgres_obj__doc__
};


/* ------------------------------------------------------------ */
/* everything for pgconn, the postgres connection object        */
/* ------------------------------------------------------------ */

typedef struct {
  PyObject_HEAD
  PGconn *conn;
  int trace;
} pgconn;

staticforward PyTypeObject pgconn_obj;
```

```
/* all that help stuff */

static char pgpy_module_documentation[]=
   "Postgres95 Python Interface";

static char pgconn_obj__doc__[]=
   "Postgres95 Connection Object";

static char pgpy_pgconn__doc__[]=
   "pgconn(host,port,dbname) = pgconn";

static char pgconn_get_dbname__doc__[]=
   "returns the name of the database you are connected to.";
static char pgconn_get_host__doc__[]=
   "returns the hostname you are connected to.";
static char pgconn_get_port__doc__[]=
   "returns the port number you are connected to.";
static char pgconn_get_options__doc__[]=
   "returns the options which were passed to the postmaster.";
static char pgconn_get_tty__doc__[]=
   "returns the tty where the postmaster logs to.";
static char pgconn_connstatus__doc__[]=
   "returns 0 if the connectio is ok., or 1 otherwise.";
static char pgconn_get_error__doc__[]=
   "returns the error that occures last.";
static char pgconn_connreset__doc__[]=
   "reset the connection to the postmaster.";
static char pgconn_switch_trace__doc__[]=
   "switch trace mode on or off.";
static char pgconn_exec_query__doc__[]=
   "executes the query passed as parameter.";


/* forward declarations for methodes */

staticforward
   pgconn  * new_pgconn(char *host, char *port, char* dbname);
staticforward void      free_pgconn(pgconn *self);
staticforward
   PyObject* pgpy_pgconn(PyObject *self,PyObject *args);
staticforward PyObject* pgconn_getattr(pgconn *self,char *name);
staticforward PyObject* pgconn_get_dbname(pgconn *self);
staticforward PyObject* pgconn_get_host(pgconn *self);
staticforward PyObject* pgconn_get_port(pgconn *self);
staticforward PyObject* pgconn_get_options(pgconn *self);
```

```
staticforward PyObject* pgconn_get_tty(pgconn *self);
staticforward PyObject* pgconn_get_error(pgconn *self);
staticforward PyObject* pgconn_connstatus(pgconn *self);
staticforward
   PyObject* pgconn_exec_query(pgconn *self,PyObject *args);
staticforward PyObject* pgconn_switch_trace(pgconn *self);
staticforward PyObject* pgconn_connreset(pgconn *self);


/* method table for class pgconn */

static struct PyMethodDef pgconn_methods[]={
  {"get_dbname"  ,(PyCFunction)pgconn_get_dbname  },
  {"get_host"    ,(PyCFunction)pgconn_get_host    },
  {"get_port"    ,(PyCFunction)pgconn_get_port    },
  {"get_options" ,(PyCFunction)pgconn_get_options },
  {"get_tty"     ,(PyCFunction)pgconn_get_tty     },
  {"connstatus"  ,(PyCFunction)pgconn_connstatus  },
  {"get_error"   ,(PyCFunction)pgconn_get_error   },
  {"connreset"   ,(PyCFunction)pgconn_connreset   },
  {"switch_trace",(PyCFunction)pgconn_switch_trace},
  {"exec_query"  ,(PyCFunction)pgconn_exec_query  },
  {NULL,NULL}
};


/* pgconn object table  */

static PyTypeObject pgconn_obj= {
  PyObject_HEAD_INIT(&PyType_Type) 0,
  "pgconn",
  sizeof(pgconn),
  0,
  (destructor)free_pgconn,
  (printfunc)0,
  (getattrfunc)pgconn_getattr,
  (setattrfunc)0,
  (cmpfunc)0,
  (reprfunc)0,
  0,0,0,
  (hashfunc)0,
  (ternaryfunc)0,
  (reprfunc)0,
  0L,0L,0L,0L,
  pgconn_obj__doc__
```

```
};



/* ------------------------------------------------------------------- */
/* everything for pgblob, the postgres binary large object interface  */
/* ------------------------------------------------------------------- */

typedef struct {
  PyObject_HEAD
  pgconn *conn;
        int fd;
        Oid blobid;
        char *buff;
} pgblob;

staticforward PyTypeObject pgblob_obj;

/* all that help stuff */


static char pgblob_obj__doc__[]=
   "Postgres95 Binary Large Object";

static char pgpy_pgblob__doc__[]=
   "pgblob(pgconn)=pgblob";

static char pgblob_bopen__doc__[]=
   "open BLOB.";
static char pgblob_bclose__doc__[]=
   "close BLOB.";
static char pgblob_bread__doc__[]=
   "read from BLOB.";
static char pgblob_bwrite__doc__[]=
   "write to BLOB.";
static char pgblob_bseek__doc__[]=
   "seek position in BLOB.";
static char pgblob_btell__doc__[]=
   "tells current position in BLOB.";
static char pgblob_bimport__doc__[]=
   "import from file to BLOB.";
static char pgblob_bexport__doc__[]=
   "export from BLOB to file.";
static char pgblob_bcreate__doc__[]=
   "create new BLOB.";
```

```
static char pgblob_bunlink__doc__[]=
   "unlink from BLOB Oid.";


/* forward declarations for methodes */

staticforward pgblob  * new_pgblob(pgconn *init);
staticforward void      free_pgblob(pgblob *self);
staticforward
   PyObject* pgpy_pgblob(PyObject *self,PyObject *args);
staticforward PyObject* pgblob_getattr(pgblob *self,char *name);
staticforward
   PyObject* pgblob_bopen(pgblob *self,PyObject *args);
staticforward PyObject* pgblob_bclose(pgblob *self);
staticforward
   PyObject* pgblob_bread(pgblob *self,PyObject *args);
staticforward
   PyObject* pgblob_bwrite(pgblob *self, PyObject *args);
staticforward
   PyObject* pgblob_bseek(pgblob *self, PyObject *args);
staticforward
   PyObject* pgblob_btell(pgblob *self);
staticforward
   PyObject* pgblob_bimport(pgblob *self,PyObject *args);
staticforward
   PyObject* pgblob_bexport(pgblob *self, PyObject *args);
staticforward
   PyObject* pgblob_bcreate(pgblob *self, PyObject *args);
staticforward PyObject* pgblob_bunlink(pgblob *self);



/* method table for class pgblob */

static struct PyMethodDef pgblob_methods[]={
  {"bopen"   ,(PyCFunction)pgblob_bopen   },
  {"bclose"  ,(PyCFunction)pgblob_bclose  },
  {"bread"   ,(PyCFunction)pgblob_bread   },
  {"bwrite"  ,(PyCFunction)pgblob_bwrite  },
  {"bseek"   ,(PyCFunction)pgblob_bseek   },
  {"btell"   ,(PyCFunction)pgblob_btell   },
  {"bimport" ,(PyCFunction)pgblob_bimport },
  {"bexport" ,(PyCFunction)pgblob_bexport },
  {"bcreate" ,(PyCFunction)pgblob_bcreate },
  {"bunlink" ,(PyCFunction)pgblob_bunlink },
```

```
  {NULL,NULL}
};


/* pgblob object table  */

static PyTypeObject pgblob_obj= {
  PyObject_HEAD_INIT(&PyType_Type) 0,
  "pgblob",
  sizeof(pgblob),
  0,
  (destructor)free_pgblob,
  (printfunc)0,
  (getattrfunc)pgblob_getattr,
  (setattrfunc)0,
  (cmpfunc)0,
  (reprfunc)0,
  0,0,0,
  (hashfunc)0,
  (ternaryfunc)0,
  (reprfunc)0,
  0L,0L,0L,0L,
  pgblob_obj__doc__
};


/* pgpylib init table, lets stick it together */

static struct PyMethodDef pgpy_methods[]={
  {"pgconn",(PyCFunction)pgpy_pgconn,1,pgpy_pgconn__doc__},
  {"pgres" ,(PyCFunction)pgpy_pgres,1,pgpy_pgres__doc__},
  {"pgblob",(PyCFunction)pgpy_pgblob,1,pgpy_pgblob__doc__},
  {NULL,NULL}
};
```

D.2 pgpylib.c

```
/* Python Postgres Interface Library
   (C) by Hans Matzen, 1997, Frankfurt, Germany

*/

/* this is pgpylib.c */

#include <stdio.h>
#include "pgpylib.h"

/* pgres constructor and destructor */

static pgres *new_pgres(pgres *init) {
  pgres *self;

  Py_Try((self=PyObject_NEW(pgres, &pgres_obj)));
  self->res=init->res;

  return self;
}

static void free_pgres(pgres *self) {
  PQclear(self->res);
  PyMem_DEL(self);
}

/* pgres methods */

static PyObject* pgres_getattr(pgres *self,char *name) {
  return Py_FindMethod(pgres_methods, (PyObject *) self,name);
}

/* returns the number of records in the result */
static PyObject* pgres_get_tuple_count(pgres *self) {
  int result;

  result=PQntuples(self->res);
  return Py_BuildValue("i",result);
 }

/* returns the number of fileds per record in the result */
static PyObject* pgres_get_field_count(pgres *self) {
  int result;
```

```
  result=PQnfields(self->res);
  return Py_BuildValue("i",result);
}

/* returns fieldname in column fieldindex */
static PyObject*
   pgres_get_fieldname_byindex(pgres *self,PyObject *args) {
  char *result;
  int fieldindex;

  Py_Try(PyArg_Parse(args,"i",&fieldindex));
  result=PQfname(self->res,fieldindex);
  return Py_BuildValue("s",result);
}

/* returns column number of fieldname */
static PyObject*
   pgres_get_fieldindex_byname(pgres *self, PyObject *args) {
  int result;
  char *fieldname;

  Py_Try(PyArg_Parse(args,"s",&fieldname));
  result= PQfnumber(self->res,fieldname);
  return Py_BuildValue("i",result);
}

/*  returns type of field with index fieldindex */
static PyObject*
   pgres_get_fieldtype(pgres *self, PyObject *args) {
  int result;
  int fieldnum;

  Py_Try(PyArg_Parse(args,"i",&fieldnum));
  result=PQftype(self->res,fieldnum);
  return Py_BuildValue("i",result);
}

/*  returns size i bytes of field with index fieldindex */
static PyObject*
   pgres_get_fieldsize(pgres *self, PyObject *args) {
  int result;
  int fieldindex;

  Py_Try(PyArg_Parse(args,"i",&fieldindex));
```

```
  result=PQfsize(self->res,fieldindex);
  return Py_BuildValue("i",result);
}

/* returns the value of field with index fieldnum
   in record tupnum     */
static PyObject*
   pgres_get_fieldvalue(pgres *self, PyObject *args) {
  char *result;
  int tupnum;
  int fieldnum;

  Py_Try(PyArg_ParseTuple(args,"ii",&tupnum,&fieldnum));
  result=PQgetvalue(self->res,tupnum,fieldnum);
  return Py_BuildValue("s",result);
}

/* returns length in bytes of value in field given
   by fieldnum, tupnum     */
static PyObject*
   pgres_get_fieldlength(pgres *self,PyObject *args) {
  int result;
  int tupnum, fieldnum;

  Py_Try(PyArg_ParseTuple(args,"ii",&tupnum,&fieldnum));
  result=PQgetlength(self->res,tupnum,fieldnum);
  return Py_BuildValue("i",result);
}

/* returns the returnvalue from the databaseserver */
static PyObject* pgres_commandstatus(pgres *self) {
  char *result;

  result=PQcmdStatus(self->res);
  return Py_BuildValue("s",result);
}

/* returns the object status of the resultobject
   helpful for errorchecking  */
static PyObject* pgres_objectstatus(pgres *self) {
 char *result;

  result=PQoidStatus(self->res);
  return Py_BuildValue("s",result);
}
```

```
/* returns an ASCII table of the whole resukt matrix */
static PyObject*
   pgres_get_resultlist(pgres *self,PyObject *args) {

  int fillAllign;
  char * fieldSep;
  int printHeader;
  int quiet;

  Py_Try(PyArg_ParseTuple(args,"isii",&fillAllign,&fieldSep,
     &printHeader,&quiet));
  PQdisplayTuples(self->res,NULL,fillAllign,
     fieldSep,printHeader,quiet);

  return Py_BuildValue("s","EOL");
}



/* pgconn constructor and destructor */

static pgconn *
   new_pgconn( char *host, char *port, char* dbname) {

  pgconn *self;

  Py_Try((self=PyObject_NEW(pgconn, &pgconn_obj)));

  self->trace=0;
  self->conn=PQsetdb(host,port,getenv("PGOPTIONS"),
     getenv("PGTTY"),dbname);

  return self;

}

static void free_pgconn(pgconn *self) {
  PQfinish(self->conn);
  PyMem_DEL(self);
}



/* pgconn methods */
```

```
static char pgconn_getdbname__doc__[]="getdbname(pgconn)";

/* returns the database name /*
static PyObject* pgconn_get_dbname(pgconn *self) {
  char *result;

  result=PQdb(self->conn);
  return Py_BuildValue("s",result);
}

/* returns the hostname of the databaseserver */
static PyObject* pgconn_get_host(pgconn *self) {
  char *result;

  result =PQhost(self->conn);
  return Py_BuildValue("s",result);
}

/* returns the portnumber of the databaseserver */
static PyObject* pgconn_get_port(pgconn *self) {
  char *result;

  result =PQport(self->conn);
  return Py_BuildValue("s",result);
}

/* returns the options set for the connection */
static PyObject* pgconn_get_options(pgconn *self) {
  char *result;

  result =PQoptions(self->conn);
  return Py_BuildValue("s",result);
}

/* returns the tty */
static PyObject* pgconn_get_tty(pgconn *self) {
  char *result;

  result =PQtty(self->conn);
  return Py_BuildValue("s",result);
}

/* returns the connectionstatus
   helpful for errorchekcing */
```

```
static PyObject* pgconn_connstatus(pgconn *self) {
  int result;

  result =PQstatus(self->conn);
  return Py_BuildValue("i",result);
}

/* if an error occured, this gets the errormessage
   from the server    */
static PyObject* pgconn_get_error(pgconn *self) {
  char *result;

  result =PQerrorMessage(self->conn);
  return Py_BuildValue("s",result);
}

/* switches tracemode on/off */
static PyObject* pgconn_switch_trace(pgconn *self) {
  if (self->trace == 1) {
    PQuntrace(self->conn);
    self->trace=0;
  } else {
    PQtrace(self->conn,stdout);
    self->trace=1;
  }

  return Py_BuildValue("i",self->trace);
}

/* resets the connection */
static PyObject* pgconn_connreset(pgconn *self) {

  PQreset(self->conn);
  return Py_BuildValue("i",0);
}

/* executes the query given by qu
   returns a pgres objecthandle   */
static PyObject*
   pgconn_exec_query(pgconn *self, PyObject *args) {

  PyObject *result;
  pgres *reshandle=malloc(sizeof(pgres)+1);
  char *qu;
```

```
  Py_Try(PyArg_Parse(args,"s",&qu));
  reshandle->res = PQexec(self->conn,qu);
  result= (PyObject *) new_pgres(reshandle);
  return result;
}

/* neede by python,  to resolve methodcalls */
static PyObject *pgconn_getattr(pgconn *self, char *name) {
  if (strcmp(name,"trace")==0) {
    return Py_BuildValue("i",self->trace);
  } else {
    return Py_FindMethod(pgconn_methods,
       (PyObject *) self,name);
  }
}


/* pgblob constructor and destructor */

static pgblob *new_pgblob( pgconn *init) {

  pgblob *self;

  Py_Try((self=PyObject_NEW(pgblob, &pgblob_obj)));

  self->blobid=0;
  self->fd=0;
  self->buff=malloc(1030);
  self->conn=init;
  Py_INCREF(init);


  return self;

}

static void free_pgblob( pgblob *self) {
    free(self->buff);
    Py_DECREF(self->conn);
    PyMem_DEL(self);
}



/* pgblob methods */
```

```
/* opens a known blob */
static PyObject* pgblob_bopen(pgblob *self, PyObject *args) {
  int result;
  int mode;

  Py_Try(PyArg_Parse(args,"i",&mode));
  result=lo_open(self->conn->conn,self->blobid,mode);
  self->fd=result;
  return Py_BuildValue("i", result);
}


/* closes a blob */
static PyObject* pgblob_bclose(pgblob *self) {
  int result;

  result =lo_close(self->conn->conn,self->fd);
  return Py_BuildValue("i",result);
}

/* reads len bytes from blob and puts them into buff */
static PyObject* pgblob_bread(pgblob *self,PyObject *args) {
  int result;
  int len;


  Py_Try(PyArg_Parse(args,("i"),&len));
  result =lo_read(self->conn->conn,self->fd,self->buff,len);

  return Py_BuildValue("i",result);
}

/* writes buff to blob returns number of written bytes */
static PyObject*
   pgblob_bwrite(pgblob *self,PyObject *args) {
  int result;
  char *buff;
  int len;

  Py_Try(PyArg_ParseTuple(args,("si"),&buff,&len));

  result =lo_write(self->conn->conn,self->fd,buff,len);

  return Py_BuildValue("i",result);
```

```
}

/* changes the position of the filepointer */
static PyObject* pgblob_bseek(pgblob *self,PyObject *args) {
  int result;
  int offset;
  int whence;

  Py_Try(PyArg_ParseTuple(args,"ii",&offset,&whence));
  result =lo_lseek(self->conn->conn,self->fd,offset,whence);
  return Py_BuildValue("i",result);
}

/* returns the position of the filepointer */
static PyObject* pgblob_btell(pgblob *self) {
  int result;

  result =lo_tell(self->conn->conn,self->fd);
  return Py_BuildValue("i",result);
}

/* export the whole blob to file fname
   be careful using this sometimes causes trouble
   with postgres   */
static PyObject*
   pgblob_bexport(pgblob *self,PyObject *args) {
  int result;
  char *fname;

  Py_Try(PyArg_Parse(args,"s",&fname));
  PQexec(self->conn->conn,"begin ");
  result =lo_export(self->conn->conn,self->blobid,fname);
  PQexec(self->conn->conn,"end;");
  return Py_BuildValue("i",result);
}


/* imports the file given by fname as blob into
   postgres, returns the blob-id */
static PyObject*
   pgblob_bimport(pgblob *self,PyObject *args) {
  Oid result;
  char *fname;

  Py_Try(PyArg_Parse(args,"s",&fname));
```

```
  PQexec(self->conn->conn,"begin ");
  result =lo_import(self->conn->conn,fname);
  PQexec(self->conn->conn,"end;");
  self->blobid=result;
  return Py_BuildValue("i",(unsigned int)result);
}


/* creates a new blob and returns the blob-id  */
static PyObject* pgblob_bcreate(pgblob *self,PyObject *args) {
    int mode;
    Oid result;

    Py_Try(PyArg_Parse(args,"i",&mode));
    result=lo_creat(self->conn->conn,mode);
    self->blobid=result;
    return Py_BuildValue("i",(unsigned int) result);
}


/* removes the blob given by blobid from the database */
static PyObject* pgblob_bunlink(pgblob *self) {
  int result;

  result=lo_unlink(self->conn->conn,self->blobid);
  return Py_BuildValue("i",result);
}

/* this is for python, for resolving methodcalls  */
static PyObject *pgblob_getattr(pgblob *self, char *name) {
  if (strcmp(name,"fd")==0) {
      return Py_BuildValue("i",self->fd);
  } else if (strcmp(name,"blobid")==0) {
      return Py_BuildValue("i",self->blobid);
  } else if (strcmp(name,"buff")==0) {
      return Py_BuildValue("s",self->buff);
  } else {
      return Py_FindMethod(pgblob_methods,
(PyObject *) self,name);
  }
}
```

```c
/* -------------------------------------------------------- */
/* pgpy module methods                                      */
/* -------------------------------------------------------- */

static PyObject* pgpy_pgres(PyObject *self,PyObject *args) {

  PyObject *result;
  pgres *p;

  Py_Try(PyArg_ParseTuple(args,"O",p));
  result = (PyObject *) new_pgres(p);

  return result;
}

static PyObject* pgpy_pgconn(PyObject *self,PyObject *args) {

  PyObject *result;
  char *host, *port, *dbname;

  Py_Try(PyArg_ParseTuple(args,"sss",&host,&port,&dbname));
  result = (PyObject *) new_pgconn(host,port,dbname);

  return result;
}

static PyObject* pgpy_pgblob(PyObject *self,PyObject *args) {

  PyObject *result;
  pgconn *c;


  Py_Try(PyArg_ParseTuple(args,"O",&c));
  result = (PyObject *) new_pgblob(c);

  return result;
}


/* -------------------------------------------------------- */
/* module pgpy init                                         */
/* -------------------------------------------------------- */
```

```
void initpgpy() {
  PyObject *m, *d;

/* create module and add functions */
  m=Py_InitModule4("pgpy",pgpy_methods,
pgpy_module_documentation,(PyObject*)NULL,PYTHON_API_VERSION);

/* these are the symbols for error reporting */
d= PyModule_GetDict(m);
ErrorObject=PyString_FromString("pgpy.error");
PyDict_SetItemString(d,"error",ErrorObject);

/* any errors ?? */

if (PyErr_Occurred()) {
  Py_FatalError("unable to initialize module pgpy");
}

}
```

Anhang E

Quellcode des Prototyps

E.1 Schnittstellenkomponente

E.1.1 Beispiel für Konfigurationsdatei

```
# ---------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname       : interface.conf
# Datum           : 03.11.1997
# letzte Änderung :
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Konfigurationsdatei des Schnittstellenelements
#
# Anmerkungen     :
# ---------------------------------------------------------------

# Portnummer fuer eingehende Auftraege
# Syntax: RECEIVE_PORT <portnr>
#
RECEIVE_PORT 6301

# Adressen der Broker
# Syntax: BROKER_ADDRESS <host>,<port>[,<timeout>]
#
BROKER_ADDRESS localhost,6401
```

```
# Python Modul in dem die Filter Routinen stehen
# Syntax: MAP_MODULE <modulname>,<modulname>,...
#
MAP_MODULE filters

# Eintraege, die einem Kommando einen Filter zuordnen
# Syntax: MAP_COMMAND <command>,<name of filterfunction>
#
MAP_COMMAND STORE,store_filter
MAP_COMMAND ACTIVATE,dummyfilter
MAP_COMMAND INFO,dummyfilter

# maximale Anzahl der offenen Auftraege im Schnittstellenobjekt
# Syntax: MAX_ORDERS <count>
#
MAX_ORDERS 10

# Logdatei
# Syntax: LOG_FILE <filename>
#
LOG_FILE ../log/ifobject.log
```

E.1.2 Hauptprogramm der Schnittstellenkomponente

```
#!/usr/users2/diplom/hans/python/python/bin/python
# ---------------------------------------------------------------
# Projekt           : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname         : ifobject.py
# Datum             : 03.11.1997
# letzte Änderung :
# Autor             : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache           : Python v1.4
# Beschreibung      : Dieses Programm realisiert die
#                     Schnittstellenkomponente des Speichersystems
#
# Anmerkungen       :
#
# ---------------------------------------------------------------

# globale Variable
CONFIG_FILE="interface.conf"

# imports
import sys
import string
import signal

# Klasse zum parsen der Konfigurationsdatei
import confparse
# Klasse zur Verwaltung einer Socketmenge
import c_multiconn
# Klasse um in ein Logfile zu schreiben
import c_log
# Klasse zur Auftragsverwaltung (Zuordnung Socket <-> Auftrag)
import c_orgalists


# Verarbeitung  Kommandozeilenparameter
# pruefe parameter
if len(sys.argv)<2:
      print "Usage: ifobject.py <name> [-f <configfile>]"
      sys.exit(1)

# hole Kommandozeilenparameter
```

```
ifobjname=sys.argv[1]
try:
      CONFIG_FILE=sys.argv[3]
except:
      pass

# installiere Signalhandler
doit=1

def sig_int(a,b):
      global doit
      # Abbruchbedingung fuer Mainloop
      doit=0

signal.signal(signal.SIGINT,sig_int)



# instanziiere Socketmengenobjekt
conn=c_multiconn.c_multiconn()

# Verarbeite Konfigurationsfile
params=confparse.confparse(CONFIG_FILE)

brlist=[]
rcvlist=[]
modlist=[]
mapdict={}
lfile="./if.log"
i=0

# durchlaufe alle Parameter und speichere sie in
# der dafür vorgesehenen Liste
while i<len(params):
      el=params[i]
      if el[0]=="RECEIVE_PORT":
        rport=eval(el[1])
        rcvlist.append(conn.open_sock("",rport))
      elif el[0]=="BROKER_ADDRESS":
        baddr=string.split(el[1],",")
        brlist.append(baddr)
      elif el[0]=="MAP_MODULE":
        mmod=el[1]
        modlist=string.split(mmod,",")
      elif el[0]=="MAP_COMMAND":
```

```
        map=string.split(el[1],",")
        mapdict[map[0]]=map[1]
      elif el[0]=="MAX_ORDERS":
        maxorders=eval(el[1])
      elif el[0]=="LOG_FILE":
        lfile=el[1]

      i=i+1


# Oeffne Logfile
log=c_log.c_log("Schnittstelle␣"+ifobjname,lfile)

log.log("Gestartet")

# init. Variablen fuer Auftragslistenverwaltung
# instanziiere Speicherliste und Jobliste
rcv_connlist=[]
jobs=c_orgalists.c_joblist()

# Schnittstellenelement-Sockets vorbereiten
i=0
log.log("Verbindung␣fuer␣Klienten␣unter:")
while (i<len(rcvlist)):
      log.log(str(conn.get_localaddr(rcvlist[i])))
      conn.listen(rcvlist[i])
      i=i+1

# ----------------------------------------------------------------
# Main Loop
# ----------------------------------------------------------------
doit=1
while doit==1:
      log.log("Warte␣auf␣Auftraege")
      active_socks=[]
      # Warte auf Auftraege
      active_socks=conn.wait_event()

      # Ereignisse abarbeiten
      i=0
      while i<len(active_socks):
        actsock=active_socks[i]

        if actsock in rcvlist:
          log.log("Ein␣Klient␣bittet␣um␣Verbindung.")
```

```
  # Klient baut verbindung auf
  newfd=conn.accept(actsock)
  rcv_connlist.append(newfd)
elif actsock in rcv_connlist:
  # Nachricht ueber bestehende Verbindung
  log.log("Ein Klient bittet um Gehoer.")
  msg=""
  msg=conn.read_sock(actsock)
  if msg=="":
    log.log("Oops, der hat einfach aufgelegt ["+
     str(actsock)+"].")
    conn.close_sock(actsock)
    del rcv_connlist[rcv_connlist.index(actsock)]
  else:
    # Verarbeitung des auftrages
    msg_parts=string.split(msg," ")
    # Nachricht filtern/konvertieren
    if mapdict.has_key(msg_parts[0]):
      # aufruf der filterroutine
      statement="import "+modlist[0]
      exec statement
      statement="msgnew="+modlist[0]+"."+
       mapdict[msg_parts[0]]+"(msg)"
      exec statement

    #verbindung zum broker aufbauen und auftrag
    # abschicken
    newfd=conn.open_sock("",0)
    conn.connect(newfd,brlist[0][0],
      eval(brlist[0][1]))
    conn.write_sock(newfd,msgnew)
    # und Auftrag in Auftragsliste speichern
    jobs.addjob(actsock,"c",0,newfd,"c")

else:
  # Nachricht ueber bestehende Verbindung
  log.log("Ein Broker bittet um Gehoer.")
  msg=conn.read_sock(actsock)
  if msg=="":
    log.log("Oops, der hat einfach aufgelegt ["+
     str(actsock)+"].")
    conn.close_sock(actsock)
    del rcv_connlist[rcv_connlist.index(actsock)]
  else:
    # zuordnung der antwort zum klienten
```

```
            clifd=jobs.get_job_by_repfd(actsock)
            tuple=jobs.get_jobdata(clifd)
            # job aus liste loeschen
            jobs.remjob(clifd)
            # verbindung zum broker abbauen
            conn.close_sock(actsock)
            # antwort an client schicken
            conn.write_sock(tuple[2],msg)
            # verbindung zum klienten beenden
            conn.close_sock(tuple[2])
            # verbindung aus liste loeschen
            del rcv_connlist[rcv_connlist.index(tuple[2])]

        i=i+1

# löschen des Verbindungsobjekts
del conn
```

E.1.3 Filterroutinen

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : filter.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Hier werden Funktionen definiert, die die
#                    eingehenden Aufträge an das interne Protokoll
#                    des Speichersystems anpassen.
# -------------------------------------------------------------
# -------------------------------------------------------------
# Name             : dummyfilter
# Datum            : 03.11.1997
# letzte Änderung :
# Typ              : Funktion
# Argumente        : msg - ein Textstring
# Rückgabewert     : msg - ein Textstring
# -------------------------------------------------------------
# Beschreibung     : Da diese Funktion nur einen Dummyfilter
#                    realisiert, wird lediglich das übergebene
#                    Argument unverändert zurückgegeben.
# -------------------------------------------------------------
def dummyfilter(msg):
        return msg

# -------------------------------------------------------------
# Name             : store_filter
# Datum            : 03.11.1997
# letzte Änderung :
# Typ              : Funktion
# Argumente        : msg - ein Textstring
# Rückgabewert     : msg - ein Textstring
# -------------------------------------------------------------
# Beschreibung     : Ersetzt das STORE Kommando durch ein
#                    PUT Kommando
# -------------------------------------------------------------
def store_filter(msg):
      return "PUT"+ msg[5:]
```

E.2 Broker

E.2.1 Beispiel für Konfigurationsdatei

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : broker.conf
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Konfigurationsdatei fuer Broker
# Anmerkungen      :
# ---------------------------------------------------------------
# Portnummer fuer eingehende Auftraege von Schnittstellenobjekten
# Syntax: RECEIVE_PORT <portnr>
#
RECEIVE_PORT 6401

# Portnummer fuer Verbindungsaufnahme der Speicher
# CONNECT_PORT <portnr>
#
CONNECT_PORT 6433

# Portnummer fuer Monitor/Steurungswerkzeug
# Syntax: CONTROL_PORT <portnr>
#
CONTROL_PORT 6444

# maximale Anzahl der offenen Auftraege im Broker
# Syntax: MAX_ORDERS <count>
#
MAX_ORDERS 50

# Logdatei
# Syntax: LOG_FILE <filename>
#
LOG_FILE ./../log/broker.log
```

E.2.2 Hauptprogramm des Brokers

```
#!/usr/users2/diplom/hans/python/python/bin/python
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : broker.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Hauptprogramm zum Broker
# Anmerkungen      :
#
# -------------------------------------------------------------

# globale Variable
CONFIG_FILE="broker.conf"

# imports
import sys
import string
import signal
import confparse
import c_multiconn
import c_log


#
# Hilfsfunktionen
#
# Loescht das Element mit dem Wert x aus der Liste Y
def dellist(y,x):
      erg=[]
      for i in range(len(y)):
        if y[i]!=x:
          erg.append(y[i])
      return erg


# Verarbeitung  Kommandozeilenparameter
# pruefe parameter
if len(sys.argv)!=2:
```

```
        print "Usage:␣broker.py␣<name>␣[-f␣<configfile>]"
        sys.exit(1)

# hole Kommandozeilenparameter und setzte Statusinformationen
# des Brokers
brokername=sys.argv[1]
brokerstate="run"
brokerqueue=0
brokercounter=0
# Wie wird verteilt automatisch=1, manuell=0
brokerdistmethod=1
# installiere Signalhandler
doit=1
def sighup(a,b):
        global brokerstate
        if brokerstate=="run":
          brokerstate="paused"
        else:
          brokerstate="run"

def sigint(a,b):
        global doit
        doit=0


signal.signal(signal.SIGHUP ,sighup)
signal.signal(signal.SIGINT ,sigint)

# instanziiere Socketmengenobjekt
conn=c_multiconn.c_multiconn()

# Verarbeite Konfigurationsfile
params=confparse.confparse(CONFIG_FILE)

# setze defaultwerte
maxorders=10
ifobj_connlist=[]
ctrl_connlist=[]
rep_connlist=[]
lfile="./broker.log"
i=0

# setzte Parameter aus Konfigurationsfile
while i<len(params):
        el=params[i]
```

```
      if el[0]=="RECEIVE_PORT":
        rport=eval(el[1])
        ifobj_connlist.append(conn.open_sock("",rport))
      elif el[0]=="CONNECT_PORT":
        cport=eval(el[1])
        rep_connlist.append(conn.open_sock("",cport))
      elif el[0]=="CONTROL_PORT":
        ctrlport=eval(el[1])
        ctrl_connlist.append(conn.open_sock("",ctrlport))
      elif el[0]=="MAX_ORDERS":
        maxorders=eval(el[1])
      elif el[0]=="LOG_FILE":
        lfile=el[1]

      i=i+1

# Oeffne Logfile
log=c_log.c_log("Broker␣"+str(brokername),lfile)
log.log("Gestartet")

# init. Variablen fuer Auftragslistenverwaltung
ifobj_list=[]
ctrl_list=[]
rep_list=[]


# instanziiere Speicherliste und Jobliste
import c_orgalists
connected_reps=c_orgalists.c_replist()
jobs=c_orgalists.c_joblist()

# initialisere Sockets
# Schnittstellenkomponenten-Sockets vorbereiten
i=0
log.log("Verbindung␣fuer␣Schnittstellenobjekte␣unter:")
while (i<len(ifobj_connlist)):
      log.log(str(conn.get_localaddr(ifobj_connlist[i])))
      conn.listen(ifobj_connlist[i])
      i=i+1

# Speicher-Sockets vorbereiten
i=0
log.log("Verbindung␣fuer␣Speicher␣unter:")
while (i<len(rep_connlist)):
      log.log(str(conn.get_localaddr(rep_connlist[i])))
```

```
      conn.listen(rep_connlist[i])
      i=i+1

# Steuerungs-Sockets vorbereiten
i=0
log.log("Verbindung fuer Managementobjekte unter:")
while (i<len(ctrl_connlist)):
      log.log(str(conn.get_localaddr(ctrl_connlist[i])))
      conn.listen(ctrl_connlist[i])
      i=i+1

# -------------------------------------------------------------
# Mainloop
# -------------------------------------------------------------
doit=1
while doit==1:
      log.log("Warte auf Auftraege")
      active_socks=[]
      # auf Ereignissen an Sockets warten
      active_socks=conn.wait_event()

      # ereignisse abarbeiten
      i=0
      while i<len(active_socks):
        actsock=active_socks[i]

        # ist der broker angehalten ?
        if brokerstate=="paused":
          for h in active_socks:
            try:

                conn.write_sock("Der Broker ist angehalten.\n")
            except:
              pass
          # while schleife verlassen
          break

        #
        # Verbindungsaufbau der verschiedenen Objekte
        #
        if actsock in rep_connlist:
          log.log("Ein Speicher bittet um Verbindung.")
          # Speicher baut verbindung auf
          newfd=conn.accept(actsock)
          rep_list.append(newfd)
```

```
if actsock in ifobj_connlist:
  log.log("Eine Schnittstelle bittet um Verbindung."
    )
  # Schnittstellenobjekt baut verbindung auf
  newfd=conn.accept(actsock)
  ifobj_list.append(newfd)

if actsock in ctrl_connlist:
  log.log("Eine Steuerinstanz bittet um Verbindung."
    )
  # M/S Werkzeug baut verbindung auf
  newfd=conn.accept(actsock)
  ctrl_list.append(newfd)

#
# Verarbeitung verschiedener Nachrichten
#

# Nachricht von einem Speicher
if actsock in rep_list:
  # Nachricht ueber bestehende Verbindung
  log.log("Ein Speicher bittet um Gehoer.")
  msg=""
  msg=conn.read_sock(actsock)
  log.log("Nachricht : "+msg[0:40])
  if msg=="":
    log.log("Oops, der hat einfach aufgelegt ["+
      str(actsock)+"].")
    conn.close_sock(actsock)
    rep_list=dellist(rep_list,actsock)
  else:
    reply=""
    parts=string.split(msg," ")
    # Anmeldung eines Speichers
    if parts[0]=="CMD_CONN":
      if connected_reps.addrep(parts[1],
        parts[2], parts[3])!=-1:
        reply="CMD_CONNCONFIRM"
      else:
        reply="CMD_CONNDENY"
    else:
      # Auftragsrueckmeldung empfangen
      jobid=jobs.get_job_by_repfd(actsock)
      njob=jobs.get_jobdata(jobid)
```

```
      # Rueckmeldung an Schnittstellenobjekt
      # schicken
      conn.write_sock(njob[2],msg)
      # Speicher als idle markieren
      connected_reps.set_idle(njob[6])
      # Auftrag aus Jobliste loeschen
      jobs.remjob(jobid)


    if reply!="":
      conn.write_sock(actsock,reply)
      log.log("Sende␣Antwort␣"+reply[0:40])

# Nachricht von einer Schnittstelle
if actsock in ifobj_list:
  # Nachricht ueber bestehende Verbindung
  log.log("Eine␣Schnittstelle␣bittet␣um␣Gehoer.")
  msg=conn.read_sock(actsock)
  log.log("Nachricht␣:␣"+msg[0:40])
  if msg=="":
    log.log("Oops,␣der␣hat␣einfach␣aufgelegt␣["+
     str(actsock)+"].")
    conn.close_sock(actsock)
    ifobj_list=dellist(ifobj_list,actsock)
  else:
    reply=""
    freerep=connected_reps.get_idlerep()
    if freerep[0]==-1:
      reply="Alle␣Speicher␣ausgelastet."
    else:
      # neuen auftrag erfassen und weiterleiten
      brokercounter=brokercounter + 1
      ns=conn.open_sock("",0)
      rep_list.append(ns)
      conn.connect(ns, freerep[2],
        eval(freerep[3]))
      jobs.addjob(actsock,"w",freerep[0],ns)
      conn.write_sock(ns,msg)

    if reply!="":
      conn.write_sock(actsock,reply)

# Nachricht von einem Managementobjekt
if actsock in ctrl_list:
  # Nachricht ueber bestehende Verbindung
```

```
log.log("Eine␣Managementinstanz␣bittet␣um␣Gehoer."
  )
msg=conn.read_sock(actsock)
log.log("Nachricht␣:␣"+msg[0:40])
if msg=="":
  log.log("Oops,␣der␣hat␣einfach␣aufgelegt␣["+
    str(actsock)+"].")
  conn.close_sock(actsock)
  ctrl_list=dellist(ctrl_list,actsock)
else:
  reply=""
  if msg=="CMD_CONFGET":
    # erstelle Konfigurationsstring
    brokerqueue=len(jobs.get_runningjobs())
    brstr="{("+ brokername + ","+
      brokerstate + ","+ str(brokercounter) +
      ","+ str(brokerqueue) +")}"
    reply=brstr + ";"+
      str(connected_reps.get_replist()) + ";"+
      str(jobs.get_runningjobs())

  elif msg=="CMD_EXIT":
    # Broker anhalten
    brokerstate="paused"
    # Alle Speicher beenden
    for i in connected_reps.get_replist():
      conn.send(i[2],"CMD_EXIT")
    # Abbruchbedingung fuer Mainloop setzen
    doit=0
  elif msg=="CMD_QUIT":
    # Notausstieg
    del conn
    raise SystemExit

  elif msg[:8]=="CMD_CHGST":
    # zustand einens speichers veraendern
    mlst=string.split(msg,"␣")

      chgid=connected_reps.get_repbyname(mlst[1])
    if chgid!=-1:
      if mlst[2]=="paused":
        # anhalten
        connected_reps.set_paused(chgid)
      else:
        # fortsetzen
```

```
              connected_reps.set_idle(chgid)
        elif msg[:9]=="CMD_MAPJOB":
          # Auftrag zu Speicher zuordnen und
          # abschicken
          mlst=string.split(msg," ")

            freerep=connected_reps.get_repinfo(mlst[2])
          # neuen auftrag erfassen und weiterleiten
          brokercounter=brokercounter + 1
          ns=conn.open_sock("",0)
          rep_list.append(ns)
          conn.connect(ns, freerep[2],
            eval(freerep[3]))
          jobs.set_repstate(actsock,"w")
          conn.write_sock(ns,msg)

        elif msg=="CMD_PAUSE":
          brokerstate="paused"
        elif msg=="CMD_CONT":
          brokerstate="run"
        elif msg="CMD_AUTO":
          brokerdistmethod=1
        elif msg="CMD_MAN":
          brokerdistmethod=0
        if reply!="":
          conn.write_sock(actsock,reply)

    i=i+1
del conn
```

E.3 Monitor-/Steuerungswerkzeug

E.3.1 Hauptprogramm des Monitor-/Steuerungswerkzeug

```
#!/usr/local/bin/python
# ---------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname       : monitor.py
# Datum           : 03.11.1997
# letzte Änderung : 08.01.1998
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Managementobjekt mit kombinierter
#                   Monitor-/Steuerungsfunktionalität
# Anmerkungen     :
#
# ---------------------------------------------------------------


# imports fuer Tk, Socketkommunikation und Stringverarbeitung
from Tkinter import *
import socket
import string


#
# Klasse des Monitor-/Steuerungswerkzeuges
#
class moni:
      #
      # Konstruktor
      #
      def __init__(self,master):
        # Variablen setzen und Socket erzeugen
        self.master=master
        self.br_anz=0
        self.sp_anz=0
        self.if_anz=0
        self.upint=5
        self.widgets={}
        self.host="localhost"
```

```
    self.port=6444
    self.connected=0
    self.conn=socket.socket(socket.AF_INET,
      socket.SOCK_STREAM)
    self.confstr=""

    # GUI aufbauen
    self.create_everything()


  #
  # Zeichnet die Oberflaeche und ordnet
  # die Ereignisse zu
  #
  def create_everything(self):
    #
    # Menues
    #
    self.mbar=Frame(self.master,relief=RAISED, bd=2)
    self.mbar.pack(side=TOP, fill=X)
    # menuebuttons einrichten
    self.m_verbindung =Menubutton(self.mbar,
      text="Verbindung",underline=0)
    self.m_broker =Menubutton(self.mbar,text="Broker",
      underline=0)
    self.m_options =Menubutton(self.mbar,text='Options',
      underline=0)
    self.m_info =Menubutton(self.mbar,text="Info",
      underline=0)

    self.m_verbindung.pack(side=LEFT)
    self.m_broker.pack(side=LEFT)
    self.m_options.pack(side=LEFT)
    self.m_info.pack(side=RIGHT)
    # menueeintraege erzeugen
    self.m1=Menu(self.m_verbindung,tearoff=NO)
    self.m1.add_command(label="Aufbauen",underline=0,
      command=self.br_connect)
    self.m1.add_command(label="Abbauen",underline=0,
      command=self.br_disconnect)
    self.m1.add_command(label="Beenden",underline=0,
      command=self.ende)

    self.m2=Menu(self.m_broker,tearoff=NO)
    self.m2.add_command(label="Anhalten",underline=0,
```

```
    command=self.br_stop)
  self.m2.add_command(label="Fortsetzen",underline=0,
    command=self.br_cont)
  self.m2.add_command(label="Manuelle␣Verteilung",
    underline=0,command=self.br_man)
  self.m2.add_command(label="Automatische␣Verteilung",
    underline=0,command=self.br_auto)


  self.m3=Menu(self.m_info,tearoff=NO)
  self.m3.add_command(label="Info",underline=0,
    command=self.info1)

  self.m4=Menu(self.m_options,tearoff=NO)
  self.m4.add_command(label="Update",underline=0,
    command=self.brupdate)
  self.m4.add_command(label="Parameter",underline=0,
    command=self.br_parms)
  # menubuttons mit menues verbinden
  self.m_verbindung.config(menu=self.m1)
  self.m_broker.config(menu=self.m2)
  self.m_options.config(menu=self.m4)
  self.m_info.config(menu=self.m3)
  # frame mit menubuttons als menueleiste einrichten
  self.mbar.tk_menuBar(self.m_verbindung, self.m_broker,
    self.m_options,self.m_info)

  # diagramm bereich einrichten
  self.f2=Frame(self.master)
  self.f2.pack()
  self.draw=Canvas(self.f2, bg='Black', height=400,
    width=500,scrollregion=(0,0,1000,400))
  self.draw.pack()
  self.dsbx=Scrollbar(self.f2,orient=HORIZONTAL)
  self.dsbx.pack(fill=X,side=BOTTOM,expand=YES)
  self.draw.config(xscrollcommand=self.dsbx.set)
  self.dsbx.config(command=self.draw.xview)

  self.draw.bind("<Double-Button-1>",self.action)

  # updateintervall selektor montieren
  self.f4=Frame(self.master)
  self.f4.pack(fill=X)

  self.f3=Frame(self.f4,relief=SUNKEN,bd=3)
```

```
    self.f3.pack(side=LEFT,fill=X)
    self.sc1=Scale(self.f3,{'from':0,'to':300},
      label="Updateintervall␣(sek.)",showvalue=YES,
      orient=HORIZONTAL,length=300,tickinterval=50,
      command=self.setupint)
    self.sc1.pack(side=LEFT)
    self.sc1.set(self.upint)

    # Status Ampel
    from ampel import ampel
    self.f5=Frame(self.f4,relief=SUNKEN,bd=3)
    self.f5.pack(side=RIGHT)
    self.amp=ampel(self.f5,20)
    self.amp.pack("side=RIGHT")
    self.amp.switch(3)
    self.amp.bind("<1>",self.connchange)
    self.lamp=Label(self.f5,text="Nicht␣verbunden")
    self.lamp.pack(side=LEFT)

    # buttons anbringen
    self.f1=Frame(self.master)
    self.f1.pack(side=RIGHT,fill=X)
    self.b4= Button(self.f1, text='Quit',
      command=self.ende)
    self.b4.pack(side=RIGHT)

# --------------------------- Ereignismethoden  --------------------

  #
  # Beendet das M/S Werkzeug
  #
  def ende(self):
    # wenn wir noch mit dem Broker verbunden sind
    if self.connected==1:
      # beim Broker abmelden
      self.br_disconnect()
    raise SystemExit

  #
  # Wertet einen Mausklicks auf die Ampel aus
  # und ruft die entsprechenden Methoden auf
  #
  def connchange(self,event):
    # ermitteln welches Ampellicht angeklickt wurde
    typ=self.amp.which(event.x,event.y)
```

```
    # gruen wurde geklickt, wenn wir noch nicht mit
    # dem Broker verbunden sind dann Verbindung aufbauen
    if typ==3:
      if self.connected==0:
        self.br_connect()
    # gelb wurde angeklickt, siehe oben
    elif typ==2:
      if self.connected==0:
        self.br_connect()
      self.brupdate()
    # rot wurde angeklickt, wenn wir mit dem Broker
    # verbunden sind dann trenne Verbindung
    else:
      if self.connected==1:
        self.br_disconnect()

  # setzt neues updateintervall, immer dann wenn
  # der schieberegler veraendert wurde
  def setupint(self,x):
    self.upint=self.sc1.get()

  #
  # ermittelt auf welche Instanz im Hauptfenster ein
  # Doppelklick ausgefuehrt wurde und zeigt das
  # entsprechende Konfigurationsfenster an.
  #
  def action(self,event):
    x=event.x
    y=event.y
    #ermittle objektname
    for i in self.widgets.keys():
      a=self.widgets[i]
      xo=a[8][0]
      yo=a[8][1]
      if (x >=xo) and (x<=xo+100) and (y>=yo) and (y<=yo+
       100):
        obj=i
    # suche objekt in configstring
    self.olist=[]
    clist=string.split(self.confstr,";")
    # ist es der Broker
    if string.split(clist[0][2:-2],",")[0]==obj:
      list=string.split(clist[0][2:-2],",")
      self.olist=["Broker",list[0],self.host,
         str(self.port),list[1],list[2],list[3]]
```

```
else:
  spl=eval(clist[1])
  for j in spl.keys():
    if spl[j][1]==obj:
      self.olist=["Speicher",spl[j][1],spl[j][2],
        spl[j][3],spl[j][4],"0","0"]
if self.olist==[]:
  afl=eval(clist[2])
  for j in afl.keys():
    if afl[j][1]==obj:
      self.olist=["Auftrag",afl[j][1],afl[j][2],
        afl[j][3],afl[j][4],"0","0"]



# fenster mit der konfiguration des objekts anzeigen
self.iw=Tk()
if1=Frame(self.iw)
if1.pack(fill=X)
if2=Frame(self.iw)
if2.pack(side=BOTTOM)
ml1=Label(if1,text="Objekttyp␣␣:␣"+self.olist[0])
ml2=Label(if1,text="Objektname␣:␣"+self.olist[1])
ml3=Label(if1,text="Hostname␣␣␣:␣"+self.olist[2])
ml4=Label(if1,text="Port␣␣␣␣␣␣␣:␣"+self.olist[3])

self.objstate=self.olist[4]
ml5=Frame(if1)

self.vcb1=StringVar()
rb1=Radiobutton(ml5,text="Laufend",value="r",
  variable=self.vcb1,command=self.sv_1)
rb2=Radiobutton(ml5,text="Angehalten",value="s",
  variable=self.vcb1,command=self.sv_2)
rb3=Radiobutton(ml5,text="Terminiert",value="t",
  variable=self.vcb1,command=self.sv_3)
rb1.pack(side=LEFT)
rb2.pack(side=LEFT)
rb3.pack(side=LEFT)
if self.objstate=="run":
  rb1.select()
  self.vcbt="run"
elif self.objstate=="paused":
  rb2.select()
  self.vcbt="paused"
```

```
    else:
      rb3.select()

    ml6=Label(if1,text="Aufträge␣␣␣:␣"+self.olist[5])
    if self.olist[0]=="Broker":
      ml7=Label(if1,text="offene␣Auftr.␣:␣"+
        self.olist[6])
    elif self.olist[0]=="Speicher":
      ml7=Label(if1,text="Auslastung␣␣␣␣:␣"+
        self.olist[6])
    else:
      ml7=Label(if1,text="")

    bl1=Button(if2,text="OK",width=10,
       command=self.setobj_parms)
    bl2=Button(if2,text="Cancel",width=10,
       command=self.iw.destroy)
    ml1.pack(anchor="w")
    ml2.pack(anchor="w")
    ml3.pack(anchor="w")
    ml4.pack(anchor="w")
    ml5.pack(anchor="w")
    ml6.pack(anchor="w")
    ml7.pack(anchor="w")
    bl1.pack(side=LEFT)
    bl2.pack(side=LEFT)
    self.iw.wait_window(self.iw)
    del self.iw


  #
  # Drei Hilfsroutinen zum auslesen
  # der Parameter des Konfigurationsfensters
  #
  def sv_1(self):
    self.vcbt="run"
  def sv_2(self):
    self.vcbt="paused"
  def sv_3(self):
    self.vcbt="term"

  #
  # sendet eine Zustandsaenderung an den Broker
  #
  def setobj_parms(self):
```

```
  # commando erstellen
  self.vcb1=self.vcbt
  self.objstate= self.vcb1
  commandstr="CMD_CHGST "+self.olist[1]+" "+
   self.objstate
  # Kommando an Broker senden
  if self.connected==1:
    self.conn.send(commandstr)
  self.iw.destroy()
  del self.olist, self.objstate, self.vcb1, self.vcbt

#
# aktualisiert die graphische Anzeige
#
def refresh(self,confstr):
  # alte widgets löschen
  for i in self.widgets.keys():
    for j in range(0,8):
      w=self.widgets[i][j]
      self.draw.delete(w)

  self.widgets={}
  self.sp_anz=0
  self.br_anz=0
  self.if_anz=0
  self.master.update_idletasks()
  # neue widgets aufbauen
  clist=string.split(confstr,";")
  # Broker erzeugen
  brlist=string.split(clist[0][2:-2],",")
  brstr="X,"+brlist[0]+","+self.host+","+str(self.port)+
   ","+brlist[1]+","+brlist[2]+","+brlist[3]
  self.broker(brstr)
  # Speicher erzeugen
  spdict=eval(clist[1])
  for i in spdict.keys():
    spl=spdict[i]
    spstr=str(spl[0])+","+spl[1]+","+spl[2]+","+spl[3]+
     ","+spl[4]+",0,0"
    self.speicher(spstr)

  ifdict=eval(clist[2])
  for i in ifdict.keys():
    ifl=ifdict[i]
```

```
#
# Zeichnet eine Auftragsinstanz
#
def auftrag(self,confstr=""):
  # Koordinaten berechnen
  y=10
  x=self.if_anz*105+5

  if confstr!="":
    clist=string.split(confstr,",")
  else:
    clist=["␣","␣","␣","␣","␣","␣","␣"]
  n=str(clist[1])
  h=str(clist[2])
  p=str(clist[3])
  s=str(clist[4])
  c=str(clist[5])

  # Auftrag zeichnen
  s0=self.draw.create_rectangle(x,y,x+100,y+100,
    fill='lightGreen')
  s1=self.draw.create_text(x+1,y+10,text="Name:"+n,
    anchor='w',width=95)
  s2=self.draw.create_text(x+1,y+25,text="Host:"+h,
    anchor='w',width=95)
  s3=self.draw.create_text(x+1,y+40,text="Port:"+p,
    anchor='w',width=95)
  s4=self.draw.create_text(x+1,y+55,text="State:"+s,
    anchor='w',width=95)
  s5=self.draw.create_text(x+1,y+70,text="Count:"+c,
    anchor='w',width=95)
  self.if_anz=self.if_anz+1
  s6=""
  s7=self.draw.create_line(x+50,y+100,self.br_anz*50+5,
    150,fill="yellow",width="5",arrow=BOTH)
  s8=(x,y)
  # Widgets merken damit man weiter darauf zugreifen
  # kann
  self.widgets[n]=(s0,s1,s2,s3,s4,s5,s6,s7,s8)

#
# zeichnet eine Speicherinstanz
#
def speicher(self,confstr=""):
  # Koordinaten berechnen
```

```
    y=290
    x=self.sp_anz*105+5

    if confstr!="":
      clist=string.split(confstr,",")
    else:
      clist=[" "," "," "," "," "," "," "]
    n=str(clist[1])
    h=str(clist[2])
    p=str(clist[3])
    s=str(clist[4])
    c=str(clist[5])
    l=str(clist[6])
    # Speicher zeichnen
    s0=self.draw.create_rectangle(x,y,x+100,y+100,
      fill='moccasin')
    s1=self.draw.create_text(x+1,y+10,text="Name :"+n,
      anchor='w',width=90)
    s2=self.draw.create_text(x+1,y+25,text="Host :"+h,
      anchor='w',width=90)
    s3=self.draw.create_text(x+1,y+40,text="Port :"+p,
      anchor='w',width=90)
    s4=self.draw.create_text(x+1,y+55,text="State:"+s,
      anchor='w',width=90)
    s5=self.draw.create_text(x+1,y+70,text="Count:"+c,
      anchor='w',width=90)
    s6=self.draw.create_text(x+1,y+85,text="Load :"+l,
      anchor='w',width=90)
    self.sp_anz=self.sp_anz+1
    s7=self.draw.create_line(x+50,y,self.br_anz*50+5,250,
      fill="white",width="5",arrow=BOTH)
    s8=(x,y)

    # Widgets merken damit man weiter darauf zugreifen
    # kann
    self.widgets[n]=(s0,s1,s2,s3,s4,s5,s6,s7,s8)

  #
  # zeichnet eine Brokerinstanz
  #
  def broker(self,confstr=""):
    # berechne Koordinaten
    y=150
    x=self.br_anz*105+5
```

```
    if confstr!="":
      clist=string.split(confstr,",")
    else:
      clist=[" "," "," "," "," "," "," "]
    n=str(clist[1])
    h=str(clist[2])
    p=str(clist[3])
    s=str(clist[4])
    c=str(clist[5])
    q=str(clist[6])
    # Zeichne Broker
    s0=self.draw.create_rectangle(x,y,x+100,y+100,
      fill='Red')
    s1=self.draw.create_text(x+1,y+10,text="Name  : "+n,
      anchor='w')
    s2=self.draw.create_text(x+1,y+25,text="Host  : "+h,
      anchor='w')
    s3=self.draw.create_text(x+1,y+40,text="Port  : "+p,
      anchor='w')
    s4=self.draw.create_text(x+1,y+55,text="State : "+s,
      anchor='w')
    s5=self.draw.create_text(x+1,y+70,text="Count : "+c,
      anchor='w')
    s6=self.draw.create_text(x+1,y+85,text="Queued: "+q,
      anchor='w')
    self.br_anz=self.br_anz+1
    s7=""
    s8=(x,y)
    # Widgets merken damit man weiter darauf zugreifen
    # kann
    self.widgets[n]=(s0,s1,s2,s3,s4,s5,s6,s7,s8)

  #
  # Zeigt das About an
  #
  def info1(self):
    iw=Tk()
    if1=Frame(iw)
    if1.pack()
    m1=Message(if1,text="
      Monitor V0.99\n\n (c) by Hans Matzen, 1997,\n
     Germany, Frankfurt/M\n email:
       hans@tm.informatik.uni-frankfurt.de",
     width="10c")
    b1=Button(if1,text="OK",command=iw.destroy)
```

```
    m1.pack()
    b1.pack()
    iw.wait_window(iw)
    del iw

  #
  # zeigt das Konfigurationsfenster fuer die Brokerparameter
  # an
  #
  def br_parms(self):
    # fenster aufbauen
    self.iw=Tk()
    self.if1=Frame(self.iw)
    self.if1.pack()
    self.if2=Frame(self.iw)
    self.if2.pack(side=BOTTOM)
    self.ml1=Label(self.if1,text="Hostname:")
    self.ml2=Entry(self.if1,width=30)
    self.ml3=Label(self.if1,text="Port:")
    self.ml4=Entry(self.if1,width=30)
    self.bl1=Button(self.if2,text="OK",width=10,
      command=self.setbr_parms)
    self.bl2=Button(self.if2,text="Cancel",width=10,
      command=self.iw.destroy)
    self.ml1.pack()
    self.ml2.pack()
    self.ml3.pack()
    self.ml4.pack()
    self.bl1.pack(side=LEFT)
    self.bl2.pack(side=LEFT)
    self.ml2.insert(END,self.host)
    self.ml4.insert(END,str(self.port))
    # warten bis Fenster geschlossen wird
    self.iw.wait_window(self.iw)


  #
  # setzt die im konfigurationsfenster gesetzten werte
  #
  def setbr_parms(self):
    self.host=self.ml2.get()
    self.port=eval(self.ml4.get())
    self.iw.destroy()
    del self.iw
```

```
#
# baut die Verbindung zum Broker auf
#
def br_connect(self):
  # pruefe ob bereits verbunden
  if self.connected==0:
    # versuche verbindung aufzubauen
    try:
      # connect durchfuehren
      self.conn.connect(self.host,self.port)
      # anzeige aktualisieren
      self.lamp.config(text="Verbunden")
      self.amp.switch(1)
      self.connected=1
      # automatische aktualisierung der Statusdaten
      # starten
      self.master.after(self.upint*1000,
        self.brupdate)
    except:
      # wenns nicht klappt
      # zeige Fehlermeldung
      import time
      self.lamp.config(text="Verbindungsfehler")
      self.amp.switch(2)
      self.master.update_idletasks()
      time.sleep(3)
      self.lamp.config(text="Nicht verbunden")
      self.amp.switch(3)

#
# fordert beim Broker die aktuellen Statusdaten an
# und zeigt sie an
#
def brupdate(self):
  # sind wir verbunden ?
  if self.connected==1:
    # wenn ja fordere statusdaten beim broker an
    self.conn.send("CMD_CONFGET")
    # warte auf antwort vom Broker
    self.confstr=self.conn.recv(8192)
    # anzeigen aktualisieren
    self.refresh(self.confstr)
    self.amp.switch(1)
    self.master.update_idletasks()
```

```
    # automatischen update starten
    self.master.after(self.upint*1000,self.brupdate)


#
# trennt die Verbindung zum Broker
#
def br_disconnect(self):
  # sind wir verbunden ?
  if self.connected==1:
    # wenn ja sende Meldung an Broker
    self.conn.send("QUIT")
    self.conn.close()
    del self.conn
    # hole neuen Socket fuer naechste Verbindung
    self.conn=socket.socket(socket.AF_INET,
      socket.SOCK_STREAM)
    self.connected=0
    # aktualisiere Anzeige
    self.lamp.config(text="Nicht␣verbunden")
    self.amp.switch(3)


#
# hält den Broker an
#
def br_stop(self):
  if self.connectd==1:
    self.conn.send("CMD_PAUSE")

#
# startet den broker wieder
#
def br_cont(self):
  if self.connectd==1:
    self.conn.send("CMD_CONT")

#
# sendet einen Zuordnungsauftrag an den Broker
#
def br_mapcommand(self,job,spname):
  if self.connectd==1:
    self.conn.send("CMD_MAPJOB␣"+str(job)+"␣"+
      str(spname))
#
```

```
      # schaltet auf manuelle verteilung um
      #
      def br_man(self):
        if self.connectd==1:
          self.conn.send("CMD_MAN")

      #
      # schaltet auf automatische verteilung um
      #
      def br_auto(self):
        if self.connectd==1:
          self.conn.send("CMD_AUTO")


#
# Hauptprogramm
#
root=Tk()
to=moni(root)
root.mainloop()
```

E.3.2 Verbindungsindikator-Widgetklasse

```
#!/usr/users2/diplom/hans/python/python/bin/python
# ----------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : ampel.py
# Datum            : 03.11.1997
# letzte Änderung : 11.11.1997
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Eine Ampelklasse zur Benutzung in graphischen
#                    Oberflaechen, die mit Tk realisiert wurde
# Anmerkungen      :
# ----------------------------------------------------------------

# imports
from Tkinter import *
import time


class ampel:

      #
      # Konstruktor
      #
      def __init__(self,master,width):
        # Variablen setzen
        self.master=master
        self.height=width*3
        self.width=width
        # Ampel aufbauen
        self.create_everything()

      #
      # Zeichnet die Ampel
      #
      def create_everything(self):
        self.pic=Canvas(self.master,background="Black",
          width=self.width,height=self.height,borderwidth=0)
        self.pic.create_rectangle(1,1,self.width+1,self.height+
         1,fill="Grey")
```

```
  ypos1 = self.width
  ypos2 = 2 * self.width
  ypos3 = 3 * self.width

  xwidth=ypos1
  if self.height<xwidth:
    xwidth=self.height

  xpos1=1
  xpos2=self.width

  self.pic.create_oval(xpos1, 1,xpos2,ypos1,tag="l1",
     fill="black")
  self.pic.create_oval(xpos1,ypos1,xpos2,ypos2,tag="l2",
     fill="black")
  self.pic.create_oval(xpos1,ypos2,xpos2,ypos3,tag="l3",
     fill="black")


#
# Ein Wrapper fuer die Standar-Tk-Pack-Function
#
def pack(self,args=""):
  self.pic.pack(side=RIGHT)

#
# Ein Wrapper fuer die Standar-Tk-Bind-Function
#
def bind(self,event,func):
  self.pic.bind(event,func)

#
# Ermittelt unter Angabe der Koordinaten
# auf welches Ampellicht geklickt wurde
#
def which(self,x,y):
  if y>2*self.width:
    self.switch(1)
    return 3
  elif y>self.width:
    self.switch(2)
    return 2
  else:
    self.switch(3)
    return 1
```

```
#
# Schaltet die Ampel um
# num =0 --> Ampel aus
# num =1 --> Ampel gruen
# num =2 --> Ampel gelb
# num =3 --> Ampel rot
#
def switch(self,num):
  if num<0 or num>3:
    return -1
  else:
    tagstr="l"+str(num)
    if num==0:
      col1="black"
      col2="black"
      col3="black"
    if num==1:
      col1="green"
      col2="black"
      col3="black"

    if num==2:
      col1="black"
      col2="yellow"
      col3="black"

    if num==3:
      col1="black"
      col2="black"
      col3="red"

    self.pic.itemconfig("l1",fill=col3)
    self.pic.itemconfig("l2",fill=col2)
    self.pic.itemconfig("l3",fill=col1)

    return 0
```

E.4 Aktiver Speicher

E.4.1 Beispiel für Konfigurationsdatei

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                      Uni-Frankfurt/M, Professur Telematik und
#                      verteilte Systeme, Prof. O. Drobnik
#                      Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : speicher.conf
# Datum            : 03.11.1997
# letzte Änderung : speicher.conf
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Konfigurationsdatei fuer Speicher
# Anmerkung        :
# -------------------------------------------------------------

#
# Portnummer fuer eingehende Auftraege von Brokern
# Syntax: RECEIVE_PORT <portnr>
#
RECEIVE_PORT 6501


#
# Portnummer fuer Verbindungsaufnahme der Broker
# BROKER_ADDRESS <hostname>,<portnr>
#
BROKER_ADDRESS localhost,6433


#
# Portnummer fuer Managementkomponente
# Syntax: CONTROL_PORT <portnr>
#
CONTROL_PORT 6544


#
# Python Modul in dem die Filter Routinen stehen
# Syntax: MAP_MODULE <modulname>
#
MAP_MODULE filters
```

```
#
# Eintraege, die einem Kommando einen Filter zuordnen
# Syntax: MAP_COMMAND <command>,<filterfunction>
#
MAP_COMMAND PUT,dummyfilter
MAP_COMMAND ACTIVATE,dummyfilter
MAP_COMMAND INFO,dummyfilter


#
# Logdatei
# Syntax: LOG_FILE <filename>
#
LOG_FILE ../log/speicher.log
```

E.4.2 Hauptprogramm des Speichers

```
#!/usr/users2/diplom/hans/python/python/bin/python
# --------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : speicher.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Dies ist das Hauptprogramm der Speicher-
#                    komponente
#
# Anmerkungen      :
#
# --------------------------------------------------------------


#
# imports
#
from sc_globals import *
import time
import select
import c_bsio
import regex
import string
import c_metadoc
import urllib
import os
import posix
import class_funcs
import pgpy
import sys
import socket
import signal
import c_log
import signalfuncs
import confparse
# Ablaufumgebungen importieren
from py_abl import *
```

```
# setzte Defaultwerte
MAXBUFF=32768
CONFIG_FILE="speicher.conf"

# verabeitung  Kommandozeilenparameter
# pruefe parameter
if len(sys.argv)<2:
      print "Usage: speicher.py <name> [-f <configfile>]"
      sys.exit(1)

# hole kommandozeilenparameter
spobjname=sys.argv[1]
try:
      CONFIG_FILE=sys.argv[3]
except:
      pass

# installiere Signalhandler
msg=""
def sig_int(a,b):
      global msg
      msg="QUIT"
signal.signal(signal.SIGUSR1,signalfuncs.sig_int)


# Verarbeite Konfigurationsfile
params=confparse.confparse(CONFIG_FILE)

brlist=[]
rcvlist=[]
ctrlist=[]
modlist=[]
mapdict={}
lfile="./sp.log"
i=0

while i<len(params):
      el=params[i]
      if el[0]=="RECEIVE_PORT":
        rport=eval(el[1])
        rcvlist.append(rport)
      elif el[0]=="BROKER_ADDRESS":
        baddr=string.split(el[1],",")
        brlist.append(baddr)
      elif el[0]=="CONTROL_PORT":
```

```
        cport=eval(el[1])
        ctrlist.append(cport)
      elif el[0]=="LOG_FILE":
        lfile=el[1]
      elif el[0]=="MAP_MODULE":
        mmod=el[1]
        modlist=string.split(mmod,",")
      elif el[0]=="MAP_COMMAND":
        map=string.split(el[1],",")
        mapdict[map[0]]=map[1]
      i=i+1


# Oeffne Logfile
log=c_log.c_log("Speicher␣"+spobjname,lfile)

log.log("Gestartet")

host="localhost"
port=rcvlist[0]

# instanziiere Auftragssocket
conn=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
try:
      conn.bind(host,port)
      conn.listen(5)
except:
      log.log("Auftragssocketadresse␣bereits␣in␣Benutzung.")
      log.log("Speicher␣beendet␣sich.")
      raise SystemExit

# Anmeldung beim Broker
conn1=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
# Verbindung aufbauen
try:
      conn1.connect(brlist[0][0],eval(brlist[0][1]))
except:
      log.log("Broker␣nicht␣verfuegbar")
      log.log("Speicher␣beendet␣sich")
      raise SystemExit

# Anmeldung schicken
conn1.send("CMD_CONN␣"+spobjname+"␣"+conn.getsockname()[0]+"␣"+
  str(conn.getsockname()[1]))
brmsg=conn1.recv(500)
```

```
# Wurde die Anmdeldung akzeptiert?
if brmsg=="CMD_CONNDENY":
      log.log("Der Broker verweigert die Anmeldung")
      raise SystemExit
else:
      log.log("Anmeldung beim Broker erfolgreich abgeschlossen."
        )


# init recv buffer da steht immer
# die letzte empfangene Nachricht drin
msg=""

# solange wir nicht beenden sollen werden Auftraege verarbeitet
while msg!="QUIT"and msg!="CMD_EXIT":
      log.log("Warte auf Auftraege an Adresse"+
       str(conn.getsockname()))
      # warten auf Nachrichten
      activ=select.select([conn.fileno()],[],[])
      # Verbindung aufbauen
      news,newsaddr=conn.accept()
      log.log("Verbindung aufgebaut mit"+str(newsaddr))

      # auf Auftrag warten
      msg=""
      msg=news.recv(MAXBUFF)
      log.log("Habe folgenden Auftrag empfangen:")
      log.log(msg)
      log.log("----End of message---")

      # konvertiere NAchricht gemaess der definierten Filter
      # im Konfigurationsfile
      msg_parts=string.split(msg," ")
      if mapdict.has_key(msg_parts[0]):
        statement="import "+modlist[0]
        exec statement
        statement="msgnew="+modlist[0]+"."+
         mapdict[msg_parts[0]]+"(msg)"
        exec statement

      # verarbeite eingegangene Nachricht
      # Auftrag parsen (Variable: msg)
      cmd=""
      params=""
```

```
tmpfname=""
tmpfile=""
pos1=regex.search("␣",msg)
if pos1>=0:
  cmd=string.upper(msg[:pos1])
else:
  cmd=string.upper(msg)

if cmd=="PUT":
  pos2=regex.search("</META>",msg)
  tmpfile=msg[pos1+1:pos2+7]
  tmpfname=class_funcs.get_uniquename()
  fd=c_bsio.c_bsio(tmpfname,"a+")
  fd.open()
  fd.write(tmpfile)
  fd.close()
  params=msg[pos2+8:]
else:
  params=msg[pos1+1:]


# ----------------------------------------------------------
# fuehre Kommando aus
# ----------------------------------------------------------

# ----------------------------------------------------------
# ein neues Dokument soll gespeichert werden
# ----------------------------------------------------------
if cmd=="PUT":
  # speichern der Metadokument Informationen
  # instanziiere Objekt zum Zugriff auf das Metadokument
  meta=c_metadoc.c_metadoc(tmpfname)
  # hole neue Dokumenten-ID
  newdocid=class_funcs.get_newdocid()
  # baue Verbindung zur Datenbank auf
  main_conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
  # ist die DTD des Dokuments schon erfasst?
  dtdnum=class_funcs.get_dtdnum(meta.get_dtd())

  # lese Metadaten aus dem Metadokument
  infostr =meta.get_info()
  infostr =infostr[string.find(infostr,">")+
   1:string.rfind(infostr,"</")]
  propdict=meta.get_property()
  methods=meta.get_methods()
```

```
# bereite SQL Query vor
dbclass="doc_description"
query="insert␣into␣"+dbclass+"␣values␣("+newdocid+",0,"

query=query + "'"+ propdict["title"] + "',"
query=query + "'"+ propdict["author"] + "',"
query=query + "'"+ propdict["owner"] + "',"
query=query + "'"+ propdict["ident"] + "',"
query=query + "'"+ propdict["pubkey"] + "',"
query=query + "'"+ infostr + "',"+str(dtdnum)+",0,0);"


# fuehre SQL Query aus (speichere Metadaten in der DB)
main_res=main_conn.exec_query(query)
erg=main_res.commandstatus()
# hat alles geklappt?
if str(erg)[0:6]!="INSERT":
  log.log("Fehler␣beim␣Speichern␣der␣Metadaten")

# Methoden in der DB speichern
for j in methods.keys():
  # SQL Query vorbereiten
  dbclass="doc_methods"
  query="insert␣into␣"+dbclass+"␣values␣("+newdocid+
   ","
  query=query + "'"+ methods[j][0] + "',"
  query=query + "'"+ methods[j][1] + "',"
  query=query + "'"+ methods[j][2] + "');"
  main_res=main_conn.exec_query(query)
  # und abschicken (eine Methode in DB sspeichern)
  erg=main_res.commandstatus()
  # hat ales geklappt ?
  if str(erg)[0:6]!="INSERT":
    log.log("Fehler␣beim␣Speichern␣der␣Methode:",
      j)

# ------------------------------
# Metadaten speichern ENDE
# ------------------------------

# Dokumenteninhalt in temporaere Datei ablegen
datafname=urllib.urlretrieve(meta.get_data()) [0]

#
# aktiviere Methode
```

```
  #
  # ermittle URL der Methode die in zum Aufruf in
  # params uebergeben werden
  if methods.has_key(params):
      methurl=methods[params][1][regex.search("URL:",
         methods[params][1])+4:]

  # hole methode
  methtmpfile,methheader=urllib.urlretrieve(methurl)

  # lasse Methode von Ablaufumgebung ausfuehren
  exec_erg=exec_mc(newdocid,methtmpfile,params,
     datafname)

  # pruefe auf fehler und sende ergebnis
  if exec_erg[0:6]=="RESULT":
    log.log("Methode␣ohne␣Fehler␣ausgefuehrt.")
  else:
    log.log("Fehler␣beim␣Ausfuehren␣der␣Methode.")
  news.send(exec_erg)



# ----------------------------------------------------------
# eine Dokumentenmethode soll aktiviert werden
# ----------------------------------------------------------
if cmd=="ACTIVATE":
  # parse Parameter
  import string
  ident=string.split(params,"␣")[0]
  method=string.split(params,"␣")[1]
  mparams=string.split(params,"␣")[2]
  # baue Verbindung zur DB auf und pruefe, ob die
  # Methode exisitiert
  pgc=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
  query="select␣*␣from␣doc_methods␣where␣docnr="+
    str(ident)+"␣and␣"
  query=query+"␣meth_class='"+method+"';"

  res=pgc.exec_query(query)
  if res.get_tuple_count()!=0:
    # ermittle Methodenname aus query Ergebnis
    methurl=res.get_fieldvalue(0,2)
    log.log("die␣Methode␣steht␣in␣",methurl)
```

```
      # hole methode
      methtmpfile,methheader=urllib.urlretrieve(methurl)

      # lasse Methode von Ablaufumgebung ausfuehren
      exec_erg=exec_mc(ident,methtmpfile,mparams)

      # pruefe auf fehler und sende ergebnis
      if exec_erg[0:6]=="RESULT":
        log.log("Methode ohne Fehler ausgefuehrt.")
      else:
        log.log("Fehler beim Ausfuehren der Methode.")
      news.send(exec_erg)

    else:
      news.send("Die Methode "+method+
        " wurde fuer das Dokument mit der ident "+str(ident)+
        " nicht gefunden.")


  # ------------------------------------------------------------
  # es sollen Informationen ueber die Programmbibliothek
  # geliefert werden
  # ------------------------------------------------------------
  if cmd=="INFO":
    import c_info
    info=c_info.c_info()
    erg=info.get_info()
    log.log("Sende Programmbibliothek Informationen.")
    news.send(erg)

  # ------------------------------------------------------------
  # der Speicher soll sich beenden
  # ------------------------------------------------------------
  if cmd=="QUIT":
    log.log("Habe Terminierungskommando erhalten.")
    news.send("Speicher beendet sich.")
    msg="QUIT"

  # ------------------------------------------------------------
  # Unbekanntes Kommando empfangen
  # ------------------------------------------------------------
  if not (cmd in ["PUT","INFO","ACTIVATE","QUIT"]):
    log.log("Kommando nicht erkannt.")
    log.log("Sende Fehlermeldung.")
    news.send("Kommando nicht erkannt.")
```

```
        log.log("Nachricht␣wurde␣verarbeitet.")

        time.sleep(2)
        log.log("Schliesse␣Socket")
        news.close()
        del news
        del newsaddr



log.log("---Speicher␣normal␣beendet---")
```

E.4.3 Filterroutinen

```
#!/usr/local/bin/python
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : filter.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Hier werden Funktionen definiert, die die
#                    eingehenden Aufträge an das interne Protokoll
#                    des Speichers anpassen.
#
# Anmerkungen      :
# -------------------------------------------------------------


# -------------------------------------------------------------
# Name             : dummyfilter
# Datum            : 03.11.1997
# letzte Änderung :
# Typ              : Funktion
# Argumente        : msg - ein Textstring
# Rückgabewert     : msg - ein Textstring
# -------------------------------------------------------------
# Beschreibung     : Da diese Funktion nur einen Dummyfilter
#                    realisiert, wird lediglich das übergebene
#                    Argument unverändert zurückgegeben.
# -------------------------------------------------------------
# Anmerkungen      :
# -------------------------------------------------------------
def dummyfilter(msg):
        return msg
```

E.4.4 Ablaufumgebung

```
#!/usr/local/bin/python
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : py_abl.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     :
#
# Anmerkungen      :
#
# ---------------------------------------------------------------
def exec_mc(docid,methodtmpfile,params,datafname=""):
      exectmpfile=class_funcs.getuniquename()
      fd1=c_bsio.c_bsio(exectmpfile,"a+")
      fd1.open()

      # schreibe skript header
      fd1.write(DL_EXECPYTHON+"\n")
      fd1.write("# Dieses Skript wurde automatisch erstellt.\n")
      fd1.write("# Digitale Bibliotheken Projekt,
        Uni-Frankfurt\n")
      fd1.write("# Lehrstuhl Telematik, 1997\n\n")
      fd1.write("from sc_globals import *\n\n")
      # schreibe globale systemvariablen
      # docid und datasource setzen
      if datafname!="":
        fd1.write("CURRENT_INFILE=\'"+datafname+"\'\n")
      fd1.write("DOCID="+str(docid)+"\n")
      fd1.write("
        # --------------- auto header ende -------------\n")

      # die Methode dranhaengen
      fd2=c_bsio.c_bsio(methodtmpfile,"r")
      fd2.open()
      buff=" "
      while buff!="":
        buff=fd2.read(512)
```

```
  fd1.write(buff)
fd2.close()
fd1.close()

# Methode ausfuehren, aufraeumen und Rueckmeldung schicken
posix.chmod(exectmpfile,0755)

if os.system(exectmpfile+"␣"+params+">"+TMPPATH+
 "output.dali")==0:
    log.log("Methode␣ohne␣Fehler␣ausgefuehrt.")
    # hole Methoden Output
    fdd=open(DL_TMPPATH+"output.dali","r")
    erg=""
    buff="␣"
    while buff!="":
      buff=fdd.readline()
      erg=erg+buff

    fdd.close()

    ret="RESULT␣"+str(docid)+"\n"+erg
    del erg,buff,fdd
else:
  log.log("Fehler␣beim␣Ausfuehren␣der␣Methode.")
  ret="ERROR␣"+str(docid)+"\n"+erg

# temporaere Dateien loeschen
os.system("rm␣"+"-f␣"+methtmpfile)
os.system("rm␣"+"-f␣"+exectmpfile)
os.system("rm␣"+"-f␣"+DL_TMPPATH+"output.dali")
# und ergebnis zurueckgeben
return ret
```

E.4.5 Klasse zur Ausführung eines INFO-Kommandos

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : c_info.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Klasse die Informationen ueber alle
#                    verfuegbaren Klassen der Programmbibliothek
#                    liefert
# Anmerkungen      :
#
# ---------------------------------------------------------------

import os
import string
from sc_globals import *



class c_info:
      #
      # Konstruktor
      #
      def __init__(self):
        self.info=""

      #
      # Liefert Informationen ueber alle Klassen im
      # Pfad der Programmbibliothek
      #
      def get_info(self):
        found=[]
        found = os.listdir(DL_MIPATH)
        miclass=[]
        h1=0
        # stelle Liste der Dateien zusammmen
        # und speichere sie in miclass
        while h1<len(found)-1:
          if found[h1][-4:-1]!=".py":
```

```
        miclass.append(found[h1])
        h1=h1+1

    # Durchlaufe alle Klassen
    while miclass !=[]:
        print "Scanne␣Klasse:␣",miclass[0]
        fd=open(DL_MIPATH+miclass[0],"r")
        zeile="␣"
        # Zeile fuer Zeile durchsuchen
        while zeile!="":
          zeile=fd.readline()
          # eine neue Klasse ?
          if string.lstrip(zeile)[0:6]=="class␣":
            self.info=self.info+zeile+"\n"
          # ein neue Methode ?
          if string.lstrip(zeile)[0:4]=="def␣":
            self.info=self.info+"␣␣␣"+zeile[:-2]+
              "\n"


        fd.close()
        del miclass[0]
        self.info=self.info+"\n\n"

    return self.info
```

E.5 Hilfsklassen

E.5.1 Logfile-Klasse

```
# ------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : c_log.py
# Datum            : 03.11.1997
# letzte Änderung  :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Eine einfache Logfile Klasse
# Anmerkungen      :
# ------------------------------------------------------------
# import Standard Dateizugriffsklasse
import c_bsio

class c_log(c_bsio.c_bsio):
      #
      # Konstruktor
      #
      def __init__(self,pname,fname):
        import posix
        # erzeuge Dateiobjekt
        c_bsio.c_bsio.__init__(self,fname,"a+")
        # Merke Logdateizeilentrailer
        self.trailer=pname+" ("+str(posix.getpid())+"): "
        # oeffne Logdatei
        self.open()
      #
      # Destruktor
      #
      def __del__(self):
        # schliesse Logdatei
        self.close()
      #
      # schreibt eine Zeile in das Logfile
      #
      def log(self,line):
        self.write(self.trailer+line+"\n")
        self.fd.flush()
```

E.5.2 Klasse zur parallelen Verwaltung einer Socketmenge (c_multiconn)

```
# -------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1997
# Dateiname       : c_multiconn.py
# Datum           : 03.11.1997
# letzte Änderung :
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Klasse zur gleichzeitigen Verwaltung
#                   mehrerer TCP/IP Verbindungen
# Anmerkungen     :
#
# -------------------------------------------------------------

# imports
import socket
import select
import ERRNO
import errno


class c_multiconn:

      #
      # Konstruktor
      #
      def __init__(self):
        # initialisiere Liste der Filedeskriptoren
        self.fdlist=[]
        # initialisiere Dictionary fuer Sockets
        self.sockdict={}

      #
      # Gibt alle vorgehaltenen Informationen ueber
      # die Sockets aus. (for debugging)
      #
      def print_lists(self):
        print "----------------------------------"
        print self.fdlist
        print self.sockdict
```

```
  print "----------------------------------"


#
# erzeugt einen neuen Socket mit der durch host, port
# gegebene Adresse, werden host, port nicht uebergeben
# wird eine freie Adresse benutzt
#
def open_sock(self,host="",port=0):
  conns=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
  try:
    conns.bind(host,port)
    fd=conns.fileno()
    self.fdlist.append(fd)
    self.sockdict[fd]=conns
    return fd
  except:
    return -1

#
# schliesst den Socket mit Nummer socknr
# und traegt ihn aus den Listen aus
#
def close_sock(self,socknr):
  sock=self.sockdict[socknr]
  try:
    sock.shutdown(1)
  except:
    sock.close()

  del sock
  self.fdlist.remove(socknr)
  del self.sockdict[socknr]
  return 1

#
# wartet auf allen Sockets auf Leseereignisse
# und gibt eine Liste der aktiven Sockets zurueck
#
def wait_event(self,sec=0):
  active=([],[],[])
  if sec==0:
    try:
      active=select.select(self.fdlist,[],[])
    except select.error:
```

```
      pass
    else:
      active=select.select(self.fdlist,[],[],sec)
    return active[0]

  #
  # schreibt eine Nachricht auf den Socket mit Nummer socknr
  # und gibt die Anzahl der geschriebenen Bytes zurueck
  #
  def write_sock(self,socknr,sendstr):
    sock=self.sockdict[socknr]
    return sock.send(sendstr)

  #
  # liest eine Nachricht vom Socket mit Nummer socknr
  # und gibt sie zurueck
  #
  def read_sock(self,socknr):
    sock=self.sockdict[socknr]
    return sock.recv(64000)


  def poll_sock(self,socknr,secs):
    pass

  #
  # baut fuer Socket mit Nummer socknr eine Verbindung
  # mit der Adresse host, port auf
  #
  def connect(self,socknr,host,port):
    sock=self.sockdict[socknr]
    return sock.connect(host,port)

  def listen(self,socknr):
    sock=self.sockdict[socknr]
    return sock.listen(5)

  #
  # wartet an Socket mit Nummer socknr auf
  # ein connect und gibt das neue Socketobjet zurueck
  #
  def accept(self,socknr):
    sock=self.sockdict[socknr]
    conns, addr =sock.accept()
    fd=conns.fileno()
```

```
    self.fdlist.append(fd)
    self.sockdict[fd]=conns
    return fd

  #
  # Gibt die lokale Adresse des Sockets mit
  # der Nummer socknr zurueck
  #
  def get_localaddr(self,socknr):
    sock=self.sockdict[socknr]
    return sock.getsockname()

  #
  # Destruktor
  #
  def __del__(self):
    # schliesse alle offenen Sockets
    for i in self.fdlist:
      self.close_sock(i)
```

E.5.3 Klassen zur Verwaltung der Speicher und der Auftraege (c_joblist, c_orgalists)

```
# ------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        :
# Datum            : 03.11.1997
# letzte Änderung  :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Hier werden zwei Klassen definiert, die
#                    die Verwaltung der angemeldeten Speicher und
#                    der laufenden Auftraege uebernehmen
# Anmerkungen      :
#
# ------------------------------------------------------------


# ------------------------------------------------------------
# Klasse zur Verwaltung der beim Broker angemeldeten Speicher
# Dabei wird zu jedem Speicher eine laufende Nummer, der Name
# des Speichers, die Adresse fuer zu sendende Auftraege
# (host,port) und der Zustand des Speichers gespeichert.
# ------------------------------------------------------------
class c_replist:

      #
      # Konstruktor
      #
      def __init__(self):
        # Variablen initialisieren
        self.count=0
        self.repdict={}
        self.IDLE="idle"
        self.BUSY="busy"
        self.PAUSED="paused"

        # Eintrag in repdict:
        # ( lfdnr, repname, hostname, port, state)

      #
      # fuegt einen neuen Speicher mit Namen name, der unter
```

```
  # host, port auf Auftraege wartet in die Liste ein.
  # Gibt die laufende Nummer innerhalb der Liste zurueck
  #
  def addrep(self,name,host,port):
    self.count=self.count+1
    self.repdict[self.count]=(self.count,name,host,port,
       self.IDLE)
    return self.count

  #
  # loescht den Speicher mit laufender Nummer repnr aus
  # der Liste
  #
  def removerep(self,repnr):
    if self.repdict.has_key(repnr):
      del self.repdict[repnr]
      return repnr
    else:
      return -1

  #
  # liefert die Nummer eines Speichers zurueck, der momentan
  # keinen
  # Auftrag verarbeitet und setzt den Zustand des Speichers
  # auf busy
  #
  def get_idlerep(self):
    i=0
    erg=(-1,"","",-1,-1)
    while i<=self.count:
      if self.repdict.has_key(i):
        nr,n,h,p,s=self.repdict[i]

        if s==self.IDLE:
          erg=(nr,n,h,p,s)
          del self.repdict[i]
          self.repdict[i]=(nr,n,h,p,self.BUSY)
          i=len(self.repdict)+1
      i=i+1
    return erg

  #
  # setzt den Zustand des Speichers mit der laufenden
  # Nummer repnr auf idle
  #
```

```
    def set_idle(self,repnr):
      if self.repdict.has_key(repnr):
        nr,n,h,p,s=self.repdict[repnr]
        del self.repdict[repnr]
        self.repdict[nr]=(nr,n,h,p,self.IDLE)
        return repnr
      else:
        return -1

    #
    # setzt den Zustand des Speichers mit der laufenden
    # Nummer repnr auf angehalten
    #
    def set_paused(self,repnr):
      if self.repdict.has_key(repnr):
        nr,n,h,p,s=self.repdict[repnr]
        del self.repdict[repnr]
        self.repdict[nr]=(nr,n,h,p,self.PAUSED)
        return repnr
      else:
        return -1

    #
    # gibt eine Liste alle in der Liste befindlichen
    # Speicher zurueck
    #
    def get_replist(self):
      return self.repdict

    #
    # sucht den Speicher mit Namen nam und gibt die repnr
    # zur"uck
    #
    def get_repbyname(self,nam):
      for i in self.repdict.keys():
        if self.repdict[i][1]==nam:
          return self.repdict[i][0]
      return -1


# imports fuer c_joblist
import string
# ----------------------------------------------------------------
# Klasse zur Verwaltung der offenen Auftraege.
```

```
# Die Groesse der Liste mu bei der Instanziierung mit angegeben
# werden, wenn sie nicht 50 sein soll.
# Die Klasse ermoeglicht die Zurordnung der Auftraege zu den
# bearbeitenden oder auftraggebenden Instanzen ueber Sockets
# ------------------------------------------------------------
class c_joblist:
      #
      # Konstruktor
      #
      def __init__(self,maxjobs=50):
        # initialisiere leere Listen und Dictionaries
        self.maxjobs=maxjobs
        self.ifdict={}
        self.repdict={}
        self.repnrdict={}
        #
        # e = empty, u =used
        #
        self.slots ="X"+ "e"* maxjobs
        #
        # e=empty, c=connected, w=waiting for reply, f=failed,
        # d=disconnected
        #
        self.ifstates ="X"+ "e"* maxjobs
        self.repstates ="X"+ "e"* maxjobs
        # clearing fd dictionaries
        for i in range(1,maxjobs+1):
          self.ifdict[i] = -1
          self.repdict[i] = -1
          self.repnrdict[i] = -1

        self.count=0

      #
      # fueht einen neuen Auftrag ein
      # if_fd   - Fildeskriptor des Sockets der
      # Schnittstellenkomponente
      # ifstate - unbenutzt
      # repnr   - laufende Nummer des Speichers aus der
      # Speicherliste
      # rep_fd  - Fildeskriptor des Sockets des Speichers der
      # den
      #           Auftrag ausfuehrt
      # repstate- Zustand des Speichers
      #
```

```
# Bei Fehler wird -1 zurueckgegeben, sonst die
# Auftragsnummer
#
def addjob(self,if_fd,ifstate,repnr,rep_fd=-1,repstate=""
  ):
  self.count=self.count+1
  if self.count>self.maxjobs:
    self.count=self.maxjobs
    return -1
  else:
    jnr=string.find(self.slots,"e")
    self.slots=self.slots[:jnr]+"u"+self.slots[jnr+1:]

    self.ifdict[jnr]=if_fd
    self.repdict[jnr]=rep_fd
    self.ifstates=self.ifstates[:jnr]+ifstate+
     self.ifstates[jnr+1:]
    self.repstates=self.repstates[:jnr]+repstate+
     self.repstates[jnr+1:]
    self.repnrdict[jnr]=repnr
    return jnr

#
# setzt den Zustand der Schnittstellenkomponente, der der
# Auftrag jobnr
# zugeordnet ist auf newstate (for debugging only)
#
def set_ifstate(self,jobnr,newstate):
  self.ifstates=self.ifstates[:jobnr]+newstate+
   self.ifstates[jobnr+1:]
  return 0

#
# setzt den Zustand des Speichers, dem der Auftrag jobnr
# zugeordnet ist auf newstate
#
def set_repstate(self,jobnr,newstate):
  self.repstates=self.repstates[:jobnr]+newstate+
   self.repstates[jobnr+1:]
  return 0

#
# loescht den Auftrag mit Nummer jobnr aus der Liste
#
def remjob(self,jobnr):
```

```
    self.slots=self.slots[:jobnr]+"e"+self.slots[jobnr+1:]

  #
  # Ermittelt den Auftrag, der der Verbindung mit
  # Filedeskriptor
  # if_fd zugeordnet ist und gibt die Nummer des Auftrags
  # zurueck.
  #
  def get_job_by_iffd(self,if_fd):
    erg=-1
    for i in self.ifdict.keys():
      if self.ifdict[i]==if_fd:
        erg=i
    return erg

  #
  # Ermittelt den Auftrag, der der Verbindung mit
  # Filedeskriptor
  # rep_fd zugeordnet ist und gibt die Nummer des Auftrags
  # zurueck.
  #
  def get_job_by_repfd(self,rep_fd):
    erg=-1
    for i in self.repdict.keys():
      if self.repdict[i]==rep_fd:
        erg=i
    return erg

  #
  # gibt ein dictionary zurueck das alle relevanten
  # Auftragsdaten
  # beinhaltet.
  #
  def get_joblist(self):
    i=0
    ergdict={}
    for i in range(1,self.maxjobs+1):
      ergdict[i]=(self.slots[i],self.ifdict[i],
         self.ifstates[i],self.repdict[i],
         self.repstates[i],self.repnrdict[jobnr])
    return ergdict

  #
  # gibt ein Dictionary zurueck, das alle laufenden
  # Auftraege beinhaltet
```

```
# in der Queue befindliche oder gestoppte Auftraege werden
# nicht
# beruecksichtigt
#
def get_runningjobs(self):
  i=0
  ergdict={}
  for i in range(1,self.maxjobs+1):
    if self.slots[i]!="e":
      ergdict[i]=(self.ifdict[i],self.ifstates[i],
         self.repdict[i],self.repstates[i])
  return ergdict

#
# gibt zu einer gegebenen jobnr alle relevanten
# Informationen
# zurueck
#
def get_jobdata(self,jobnr):
  return (jobnr,self.slots[jobnr],self.ifdict[jobnr],
     self.ifstates[jobnr],self.repdict[jobnr],
     self.repstates[jobnr],self.repnrdict[jobnr])
```

E.5.4 Funktionen zum parsen der Konfigurationsdateien

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : confparse.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Funktion zum parsen der Konfigurationsdateien
#                    gibt eine Liste von Listen der gueltigen
#                    Zeilen der Datei zurueck
# Anmerkungen      :
#
# ---------------------------------------------------------------
def confparse(fname):
      import string
      parameters=[]

      # durchlaufe alle Zeilen der Konfigurationsdatei fname
      for line in open(fname,"r").readlines():
        # sortiere Kommentare und leere Zeilen aus
        if line[0]!="#"and len(line)>1:
          # jede Zeile in einer Liste ablegen
          llist=string.split(line)
          # und an die Parameterliste anhaengen
          parameters.append(llist)
      # Ergebnisliste zurueckgeben
      return parameters
```

E.6 Programmbibliothek für Dokumentenmethoden

E.6.1 Basisklassen und Funktionen

Klasse c_bsio

```
#!/usr/local/bin/python
# ---------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname       : c_bsio.py
# Datum           : 03.11.1997
# letzte Änderung :
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Eine einfache Klasse, die Methoden für
#                   Dateizugriffe bereitstellt
# Anmerkungen     :
#
# ---------------------------------------------------------------


class c_bsio:
      # Dateiname und Filedekriptor init.
      fname=""
      fd=0

      #
      # Konstruktor
      # Initialisiert die Variablen
      # filename - Dateiname
      # mode     - Modus wie unter 'man fopen' beschrieben
      #
      def __init__(self,filename,mode="rw"):
        self.fname=filename
        self.mode=mode


      #
      # Öffnet die Datei
```

```
# gibt den Filedeskriptor
# der geöffneten Datei zurück
#
def open(self):
  self.fd=open(self.fname,self.mode)
  return self.fd


#
# Schliesst die Datei
#
def close(self):
  return self.fd.close()



#
# liest size Bytes aus der Datei und gibt sie zurueck
#
def read(self,size):
  return self.fd.read(size)


#
# Liest solange aus der Datei bis entweder
# ein Zeilenumbruch oder das Dateiende erreicht ist
#
def readline(self):
  return self.fd.readline()


#
# Schreibt einen String in die Datei
# und gibt die Anzahl der  geschriebenen Bytes zurueck

def write(self,buff):
  return self.fd.write(buff)


#
# Setzt den Dateizeiger auf eine neu Position
# Argumente       : offset - die Anzahl der Bytes um die
# der
# Dateizeiger bzgl. der von whence bestimmten
# Position verschoben werden soll.
```

```
    # whence - 0 für den Anfang der Datei,
    #          1 für die aktuelle Position
    #          2 für das Ende der Datei

    def seek(self,offset,whence):
      return self.fd.seek(offset,whence)


    #
    # Ermittelt die aktuelle Position des Dateizeigers
    #

    def tell(self):
      return self.fd.tell()
```

Klasse c_sgmlread

```
# --------------------------------------------------------------
# Projekt           : Digitale Bibliotheken Projekt
#                      Uni-Frankfurt/M, Professur Telematik und
#                      verteilte Systeme, Prof. O. Drobnik
#                      Diplomarbeit, Matzen,Hans, 1998
# Dateiname         : c_sgmlread.py
# Datum             : 09.12.1997
# letzte Änderung :
# Autor             : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache           : Python v1.4
# Beschreibung      : Klasse zum komfortablen Zugriff auf die
#                     Elemente eines als Bitstrom vorliegenden
#                     SGML-Dokuments
# Anmerkungen       : Diese Klasse wird von der Klasse c_bsio
#                     abgeleitet.
# --------------------------------------------------------------

# imports
import regex
import string
import c_bsio

class c_sgmlread(c_bsio.c_bsio):
      #
      # Konstruktor
      # fname gibt den Dateinamen des SGML-Dokuments an
      #
      def __init__(self,fname):
        # Konstruktor der Basisklasse
        c_bsio.c_bsio.__init__(self,fname,"r")
        # Oeffnen der Eingabedatei
        self.open()

      #
      # Destruktor
      #
      def __del__(self):
        # Datei schliessen
        self.close()
      #
      # Lese-/Schreibmarke auf den Dateianfang setzen
      #
      def seek_begin(self):
```

```
  self.fd.seek(0,0)

#
# Lese-/Schreibmarke auf das Dateiende setzen
#
def seek_end(self):
  self.fd.seek(0,2)

#
# liest das Doctype Tag des SGML-Dokuments aus
# und gibt es zurueck
#
def get_doctype(self):
  self.oldpos=self.tell()
  self.seek(0,0)
  buff=" "
  erg=""
  while regex.search("<!DOCTYPE",buff)==-1 and buff!="":
    buff=self.fd.readline()

    if buff =="":
      erg= ""
    else:
      self.fd.seek(-len(buff),1)
      buff=""
      while buff!=">"and buff!="[":
        buff=self.read(1)
        erg=erg+buff

    if len(erg)>0 and erg[len(erg)-1]=="[":
      erg=erg[0:-2]
      erg=erg+">"
    self.seek(self.oldpos,0)
    return erg

#
# liefert den DTD Bezeichner des SGML-Dokuments zurueck
#
def get_dtdname(self):
  dtyp=self.get_doctype()
  return string.split(dtyp[1:-1])[1]
#
# liefert den Filenamen der DTD zurueck
#
def get_dtdfile(self):
```

```
    dtyp=self.get_doctype()
    return string.split(dtyp[1:-1])[3][1:-1]
  #
  # gibt eine Liste alle Attribute des Doctype-Tags zurueck
  #
  def get_dtd(self):
    import string
    doctypetag=self.get_doctype()
    doctypetag=doctypetag[2:-1]
    return string.split(doctypetag,"␣")

  #
  # sucht ausgehend von der aktuellen Position der
  # Lese-/Schreibmarke das naechste Tag mit Namen str
  #
  def get_tag(self,str):
    buff="␣"
    while regex.search("<"+str+"*",buff)==-1 and
        regex.search("</"+str+"*",buff)==-1 and buff!="":
      buff=self.fd.readline()
      if buff!="":
        self.fd.seek(-len(buff)+regex.search("<"+str,
            buff)-1,1)
        if regex.search("</"+str+"␣",buff)!=-1 or
            regex.search("</"+str+">",buff)!=-1:
          self.fd.seek(1,1)
          erg=""
          buff=""
          while buff!=">":
            buff=self.fd.read(1)
            erg=erg+buff
      else:
        erg=""

    return erg

  #
  # sucht ausgehend von der aktuellen Position der
  # Lese-/Schreibmarke das naechste Tag
  #
  def get_nexttag(self):
    buff="␣"
    while regex.search("<",buff)==-1 and buff!="":
      buff=self.fd.readline()
```

```
    if buff!="":
      self.fd.seek(-len(buff)+regex.search("<",buff),1)
      # lese Taginhalt
      erg=""
      buff=""
      while buff!=">":
        buff=self.fd.read(1)
        erg=erg+buff
    else:
      erg=""
    return erg

  #
  # sucht ausgehend von der aktuellen Position der
  # Lese-/Schreibmarke das naechste Tag mit Namen str
  # und liefert den Inhalt des Tags zurueck
  #
  def get_tagcontents(self,str):
    erg=self.get_tag(str)
    if erg=="</"+str+">":
      erg=self.get_tag(str)
    erg1=""
    buff="␣"
    while regex.search("</"+str+"␣",buff)==-1 and
      regex.search("</"+str+">",buff)==-1 and buff!="":
      buff=self.fd.readline()
      erg1=erg1+buff

    erg1=erg1[:regex.search("</"+str+">",erg1)+len(str)+3]
    return erg+erg1

  #
  # teilt ein gegebenes Tag (tagstr) in den Tagnamen
  # und die einzelnen Attribute auf und liefert ein
  # Dictionaries zurueck, dessen Schluessel die
  # Attributnamen sind.
  #
  def split_tag(tagstr):
    ergdict={}

    # tagklammern entfernen
    tagstr=tagstr[1:-1]

    #hole tagname
    pos=string.find(tagstr,"␣")
```

```
tag=tagstr[:pos]
tagstr=tagstr[pos:]

# tagname in dictionary einfuegen
ergdict["TAG"]=tag

# hole attribute
while len(tagstr)!=0:

  # whitespace entfernen
  tagstr=string.strip(tagstr)

  # attributname holen
  pos1 = string.find(tagstr,"=")
  attrib = string.strip( tagstr[:pos1] )
  tagstr = tagstr[pos1+1:]

  # whitespace entfernen
  tagstr=string.strip(tagstr)

  # hole werte
  pos3 = string.find(tagstr,"␣")
  if pos3 == -1:
    pos3=len(tagstr)
  wert = string.strip( tagstr[:pos3] )
  tagstr = tagstr[pos3+1:]

  # attribute werte paar in dictionary einfuegen
  ergdict[attrib]=wert

return ergdict
```

Verwaltungsfunktionen

```
# -----------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : class_funcs.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Hier stehen verschiedene Funktionen, die von
#                    den Paketen verwendet werden.
# Anmerkungen      :
#
# -----------------------------------------------------------------

# standard imports
import time
# globale Variable holen
from sc_globals import *

# Postgres interface laden
import pgpy

#
# gibt eine neue noch nicht benutzte lokale docid zurueck
#
def get_newdocid():
        conn = pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        if conn.connstatus() ==0:
            query="select docnr from local_doc_ids;"
            res = conn.exec_query(query)
            docid = res.get_fieldvalue(0,0)
            newdid = eval(docid) + 1
            query1="update local_doc_ids set docnr="+
              str(newdid)+";"
            res1 = conn.exec_query(query1)
            return docid
        elif conn.connstatus() == 1:
            return -1
#
# gibt eine neue dtd idnr zurueck
#
```

```
def get_newdtdid():
        conn = pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        if conn.connstatus() ==0:
            query="select␣dtdnr␣from␣local_doc_ids;"
            res = conn.exec_query(query)
            docid = res.get_fieldvalue(0,0)
            newdid = eval(docid) + 1
            query1="update␣local_doc_ids␣set␣dtdnr="+
              str(newdid)+";"
            res1 = conn.exec_query(query1)
            return docid
        elif conn.connstatus() == 1:
            return -1


# erzeugt einen eindeutigen temporaeren Dateinamen
def get_uniquename():
        time.sleep(1)
        return str(int(time.time()))

#
# ermittelt die dtdnr einer gegebenen doctype beschreibung,
# exisitert die dtd nicht in der db, so wird sie angelegt
#
def get_dtdnum(lst):
      dtdid=lst[1]
      dtdref=lst[2]
      dtddesc=lst[3]

      conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
      query="select␣dtdnr␣from␣dtd␣where␣dtdid='"+dtdid+"';"
      res=conn.exec_query(query)

      if res.get_tuple_count()==0:
        newid=get_newdtdid()
        query="insert␣into␣dtd␣values␣("+newid+",'"+dtdid+"',"
        if dtdref=="SYSTEM":
          query=query+"'','"+dtddesc+"');"
        else:
          query=query+"'"+dtddesc+"','');"

        res=conn.exec_query(query)
        if res.commandstatus()[0:6] != "INSERT":
          print "FEHLER␣␣bei␣Insert␣in␣dtd."
```

```
        return newid
      else:
        return res.get_fieldvalue(0,0)

#
# ermittelt den dtd filename einer gegebenen doctype beschreibung,
# exisitert die dtd nicht in der db, so wird -1 zurueck gegeben
#
def get_dtdfile(lst):
      dtdid=lst[1]
      dtdref=lst[2]
      dtddesc=lst[3]

      conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
      query="select * from dtd where dtdid='"+dtdid+"';"

      res=conn.exec_query(query)


      if res.get_tuple_count()!=0:
         return res.get_fieldvalue(0,3)
      else:
        return -1
```

E.6.2 Paket Bitstream (p_bitstream.py)

```
# ------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : p_bitstream.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Paket Bitstream, fasst alle Bitstrompaket
#                    Klassen zusammen
#
# Anmerkungen      :
#
# ------------------------------------------------------------
import c_store
import c_bsio
import c_search
import c_glimpsearch
```

Klasse c_store

```
# ------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_store.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Klasse c_store bietet Methoden, um
#                    Dokumente in Form eines Bitstroms im
#                    Speicher abzulegen.
# Anmerkungen      : Die Variablen CURRENT_INFILE und NEWDOCID
#                    werden von der Ablaufumgebung gesetzt.
#
# ------------------------------------------------------------
```

```
#imports

import sc_globals
import class_funcs
CURRENT_INFILE="␣"
NEWDOCID=0

class c_store:

        infile=""
        outfile=""
        docid=0
        fd_in=0
        fd_out=0
        #
        # Konstruktor, setzt alle notwendigen Variablen
        # CURRENT_INFILE und NEWDOCID, werden von der
        # Ablaufumgebung gesetzt
        #
        def __init__(self):
            self.infile=sc_globals.DL_INPATH+
             CURRENT_INFILE
            #self.docid=class_funcs.get_newdocid()
            self.docid=NEWDOCID
            if self.docid== -1:
                print "Bekomme␣keine␣neue␣Doc␣Id."
                raise KeyboardInterrupt
            self.outfile=sc_globals.DL_BSOUTPATH+
             str(self.docid)+".bs"

        #
        # oeffnet den Eingabebitstrom
        #
        def open_in(self):
            self.fd_in=open(self.infile,"r")

        #
        # oeffnet den Ausgabebitstrom
        #
        def open_out(self):
            self.fd_out=open(self.outfile,"a+")
        #
        # schliesst den Eingabebitsrom
        #
        def close_in(self):
```

```
        self.fd_in.close()
    #
    # schliesst den Ausgabebitstrom
    #
    def close_out(self):
        self.fd_out.close()
    #
    # liest size Bytes aus dem Eeingabebitstrom
    #
    def read_in(self,size):
        return self.fd_in.read(size)
    #
    # schreibt outstr in den Ausgabebitstrom
    #
    def write_out(self,outstr):
        self.fd_out.write(outstr)

    #
    # setzt die Position der Lese-/Schreibmarke
    # des Eingabebitstroms Syntax: wie Python seek Befehl
    #
    def seek_in(self,offset,whence):
        return self.fd_in.seek(offset,whence)

    #
    # setzt die Position der Lese-/Schreibmarke
    # des Ausgabebitstroms Syntax: wie Python seek Befehl
    #
    def seek_out(self,offset,whence):
        return self.fd_out.seek(offset,whence)
    #
    # liefert die Position der Lese-/Schreibmarke
    # des Eingabebitstroms zurueck
    #
    def tell_in(self):
        return self.fd_in.tell()

    #
    # liefert die Position der Lese-/Schreibmarke
    # des Ausgabebitstroms zurueck
    #
    def tell_out(self):
        return self.fd_out.tell()

    #
```

```
# vermerkt in den Metadaten des Dokuments, dass es als
# Bitstrom im Speicher abgelegt wurde.
#
def register_store(self):
  query="
    update␣doc_description␣set␣bs_stored='t'␣where␣docnr="
  query=query+str(self.docid)+";"
  res=self.conn.exec_query(query)
  if res.commandstatus()[0:6]!="UPDATE":
    print "Fehler␣bei␣Update␣in␣register_sgmlstore"

    return -1
  else:
    return 0
```

Klasse c_search

```
# -------------------------------------------------------------
# Projekt           : Digitale Bibliotheken Projekt
#                      Uni-Frankfurt/M, Professur Telematik und
#                      verteilte Systeme, Prof. O. Drobnik
#                      Diplomarbeit, Matzen,Hans, 1998
# Dateiname         : c_search.py
# Datum             : 09.12.1997
# letzte Änderung : 15.01.98
# Autor             : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache           : Python v1.4
# Beschreibung      : Klasse zum Volltextsuche in einem bestimmten
#                     Dokument
# Anmerkungen       :
#
# -------------------------------------------------------------

class c_search:
      #
      # Konstruktor, initialisert Variable
      #
      def __init__(self,docid):
        from sc_globals import *
        self.dir=DL_FULLTEXT_DIR
        self.docid=docid

      #
      # Durchsucht das Dokument nach dem durch searchstr
      # gegebenen und/oder-verkn"upften␣Begriffen
      # Dabei steht ";␣fuer␣oder" und "&␣fuer␣und"
      # Ergebnis ist eine Liste von Zeilen, in denen
      # der Ausdruck vorkommt
      #
      def search(self,searchstr):
        slist=string.split(searchstr,"&")
        erglist=[]
        for i in slist:
          erglist.append(self.searchor(i))

        erg=[]
        for i in erglist[0]:
          flag=1
          for j in erglist[1:]:
            if not (i in j):
```

```
          flag=0
          break
      if flag==1:
        erg.append(i)

    return erg

  #
  # Wei search nur werden lediglich veroderter
  # Ausdruecke verabreitet
  #
  def searchor(self,searchstr):
    slist=string.split(searchstr,";")
    fd=open(self.dir+self.docid,"r")
    buff="␣"
    while buff!="":
      buff=fd.readline()
      for i in slist:
        if string.find(buff,i):
          erg.append(buff)
    fd.close()
    return erg
```

Klasse c_glimpsearch

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_glimpsearch.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Eine Klasse um mit glimpse eine Volltext-
#                    recherche durchzufuehren.
# Anmerkungen      :
#
# ---------------------------------------------------------------

class c_glimp_search:
      #
      # Konstruktor, initialisert Variable
      #
      def __init__(self,sparams):
        import string
        from sc_globals import *
        self.dir=DL_FULLTEXT_DIR
        self.params=sparams
        self.resfile=DL_FULLTEXT_RESULTS

      #
      # Durchsucht alle als Bitstrom gespeicherten Dokumente
      # nach dem durch searchstr
      # gegebenen und/oder-verkn"upften␣Begriffen
      # Dabei steht ";␣fuer␣oder" und "&␣fuer␣und"
      # Ergebnis ist eine Liste von Zeilen, in denen
      # der Ausdruck vorkommt
      #
      def glimp_search(self,searchstr):
        slist=string.split(searchstr,"&")
        erglist=[]
        for i in slist:
          erglist.append(self.glimp_searchor(i))

        erg=[]
        for i in erglist[0]:
```

```
      flag=1
      for j in erglist[1:]:
        if not (i in j):
          flag=0
          break
      if flag==1:
        erg.append(i)

    return erg

  #
  # Wie glimp_search es werden aber lediglich veroderter
  # Ausdruecke verarbeitet
  #
  def glimp_searchor(self,searchstr):
    import os
    erg=[]
    os.system("glimpse␣-y␣-H␣"+self.dir+"␣"+self.params+"␣"
       +searchstr+"␣>"+self.resfile)
    fd=open(self.resfile,"r")
    buff="␣"
    while buff!="":
      buff=fd.readline()
      erg.append(buff[:-1])

    fd.close()
    os.unlink(self.resfile)
    return erg[:-1]
```

E.6.3 Paket SGML (p_sgml.py)

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : p_sgml.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Paket SGML, fasst alle SGML-Paket Klassen
#                    zusammen
#
# Anmerkungen      :
#
# ---------------------------------------------------------------
import c_sgmlquery
import c_sgmlread
import c_sgmlsearch
import c_sgmlstore
```

Klasse c_sgmlquery

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_sgmlquery.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Eine Klasse die SGML spezifische Abfragen auf
#                    einem bestimmten Dokument durchfuehrt.
# Anmerkungen      :
# ---------------------------------------------------------------

# imports
from sc_globals import *
import pgpy
```

```
class c_sgmlquery:
      #
      #  Konstruktor
      #
      def __init__(self,ident):
        self.ident=ident
        # Datenbankverbindungsobjekt instanziieren
        self.pgconn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)

      #
      # liefert das tagcount'e Tag mit Namen tagname zurueck
      #
      def sgmlquery_tag(self,tagname,tagcount):
        query="select␣*␣from␣doc_tags␣where␣docnr="+
          str(self.ident)
        query=query + "␣and␣contents␣ *␣'"+tagname+
          "*'␣order␣by␣elementnr;"

        res=self.pgconn.exec_query(query)
        if res.get_tuple_count()>0:
          return [ res.get_fieldvalue(tagcount-1,
             res.get_fieldindex_byname("contents")),
             res.get_fieldvalue(tagcount-1,
             res.get_fieldindex_byname("elementnr")) ]
        else:
          return ["",0]
      #
      # liefert den Inhalt des tagcount'en Tags
      # mit Namen tagname zurueck
      #
      def sgmlquery_tagcontents(self,tagname,tagcount):
        pos1=self.sgmlquery_tag(tagname,tagcount)[1]
        query1="select␣contents,
           elementnr␣from␣doc_tags␣where␣docnr="+str(self.ident)
        query1=query1 + "␣and␣elementnr>"+str(pos1)
        query1=query1 +"and␣contents␣ *␣'</"+tagname+
          "'␣order␣by␣elementnr;"
        res1=self.pgconn.exec_query(query1)
        if res1.get_tuple_count>0:
          pos2=res1.get_fieldvalue(0,1)

          query2= "select␣contents,
             elementnr␣from␣doc_elements*␣where␣docnr="
          +str(self.ident)
```

```
      query2=query2+"␣and␣elementnr>="+str(pos1)
      query2=query2+"␣and␣elementnr<="+str(pos2)+
        "␣order␣by␣elementnr;"

      res2=self.pgconn.exec_query(query2)
      if res2.get_tuple_count>0:
        j=res2.get_tuple_count()
        i=0
        erg=""
        while i < j:
          erg=erg+str(res2.get_fieldvalue(i,0))+"\n"
          i=i+1

        return erg[:-1]
      else:
        return ""
    else:
      return ""

  #
  # liefert den Strukturbaum als ASCII Text zurueck
  #
  def sgmlquery_structure(self):
    query="select␣contents,
       elementnr␣from␣doc_tags␣where␣docnr="+ str(self.ident) +
       "␣order␣by␣elementnr;"

    res=self.pgconn.exec_query(query)
    j=res.get_tuple_count()
    i=0
    erg=""
    while i < j:
      erg=erg+str(res.get_fieldvalue(i,0))+"\n"
      i=i+1
    return erg[:-1]

  #
  # gibt das gesamte Dokument zurueck
  #
  def sgmlquery_doc(self):
    query="select␣contents,
       elementnr␣from␣doc_elements*␣where␣docnr="+str(self.ident)+
      "␣order␣by␣elementnr;"

    res=self.pgconn.exec_query(query)
```

```
    j=res.get_tuple_count()
    i=0
    erg=""
    while i < j:
      erg=erg+str(res.get_fieldvalue(i,0))+"\n"
      i=i+1
    return erg

  #
  # gibt die Anzahl der Tags mit Namen tagname zurueck
  #
  def sgmlquery_tagcount(self,tagname):
    query="select contents from doc_tags where docnr="+
      str(self.ident)+" and contents  * '"+tagname+
      "';"
    res=self.pgconn.exec_query(query)
    return res.get_tuple_count()

  #
  # teilt ein gegebenes Tag (tagstr) in den Tagnamen
  # und die einzelnen Attribute auf und liefert ein
  # Dictionaries zurueck, dessen Schluessel die
  # Attributnamen
  # sind.
  #
  def sgmlquery_splittag(self,tagstr):
    import string

    ergdict={}
    t=""
    for j in range(len(tagstr)):
      if tagstr[j]!="\012":
        t=t+tagstr[j]

    tagstr=t

    # tagklammern entfernen
    tagstr=tagstr[1:-1]

    #hole tagname
    pos=string.find(tagstr," ")
    tag=tagstr[:pos]
    tagstr=tagstr[pos:]

    # tagname in dictionary einfuegen
```

```
ergdict["TAG"]=tag

#hole attribute
while len(tagstr)!=0:

  # whitespace entfernen
  tagstr=string.strip(tagstr)

  # attributname holen
  pos1 = string.find(tagstr,"=")
  attrib = string.strip( tagstr[:pos1] )
  tagstr = tagstr[pos1+1:]

  # whitespace entfernen
  tagstr=string.strip(tagstr)

  # hole werte
  if len(tagstr)>0:
    if tagstr[0]=="\'":
      tagstr=tagstr[1:]
      pos3 = string.find(tagstr,"\'")
    else:
      pos3 = string.find(tagstr,"␣")
  else:
    pos3 = string.find(tagstr,"␣")

  if pos3 == -1:
    pos3=len(tagstr)
  wert = string.strip( tagstr[:pos3] )

  tagstr = tagstr[pos3+1:]

  # attribute werte paar in dictionary einfuegen
  ergdict[attrib]=wert

return ergdict
```

Klasse c_sgmlsearch

```
# ---------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1998
# Dateiname       : c_sgmlsearch.py
# Datum           : 09.12.1997
# letzte Änderung : 15.01.1998
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Dies Klasse realisiert ein Suchinterface fuer
#                   in der Datenbank abgelegte SGML Dokumente
# Anmerkungen     :
#
# ---------------------------------------------------------------

# imports
import pgpy

class c_sgmlsearch:
      #
      # Konstruktor
      #
      def __init__(self):
        # Verbindungsobjekt zur Datenbank instanziieren
        self.conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)

      #
      # Durchsucht alle in der Datenbank gespeicherten
      # Tags nach searchstr und findet auch aehnliche
      # Ausdruecke.
      # Als Ergebnis wird eine ASCII Liste von Dokumenten ID’s
      # zurueckgegeben
      #
      def sgmlsearch_tag(self,searchstr):
        # SQL Abfrage zusammenbauen
        query="select␣*␣from␣doc_tags␣where␣contents␣ *"+
         searchstr+";"
        # Abfrage ausfuehren
        res=self.conn.exec_query(query)
        # Fehler aufgetreten ?
        if res.get_tuple_count()==0:
          # wenn ja, -1 zurueckgeben
```

```
      return -1
    else:
      #wenn nein Ergebnis zurueckgeben
      erg=res.get_resultlist(1,"|",1,0)
      return erg

  #
  # Durchsucht die Text Klasse der Datenbank
  # nach searchstr und findet auch aehnliche
  # Ausdruecke.
  # Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
  # zurueckgegeben
  #
  def sgmlsearch_text(self,searchstr):
    # SQL Abfrage zusammenbauen
    query="select * from doc_text where contents  *"+
     searchstr+";"
    # Abfrage ausfuehren
    res=self.conn.exec_query(query)
    # Fehler aufgetreten ?
    if res.get_tuple_count()==0:
      # wenn ja, -1 zurueckgeben
      return -1
    else:
      #wenn nein Ergebnis zurueckgeben
      erg=res.get_resultlist(1,"|",1,0)
      return erg

  #
  # virtuelle Methode, muss in einer abgeleiteten
  # Klasse ueberschrieben werden und ist zum Durchsuchen
  # von Binaerdaten gedacht ;-)
  #
  def sgmlsearch_bin(self):
        pass

  #
  # Durchsucht die Tag- und die Text- Klasse der Datenbank
  # nach searchstr und findet auch aehnliche
  # Ausdruecke.
  # Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
  # zurueckgegeben.
  #
  def sgmlsearch_doc(self,searchstr):
    erg=[]
```

```
    erg1=self.sgmlsearch_tag(searchstr)
    erg2=self.sgmlsearch_text(searchstr)
    if erg1!=-1:
      erg.append(erg1)
    if erg2!=-1:
      erg.append(erg2)
    return erg

  #
  # ermittelt die Dokumenten-ID's aller Dokumente, die der
  # in dtdid gegebenen DTD entsprechen. Dabei kann dtdid
  # die Public- oder die System-ID einer DTD angeben.
  # Das Ergebnis wird in Form einer Lsite zur"uckgeliefert.
  #
  def sgmlsearch_dtddocs(self,dtdid):
    # DTD-Nummer ermitteln
    query="select␣dtdnr␣from␣dtd␣where␣publicname␣="+
      dtdid + "or␣systemname␣="+dtdid+";"
    # Abfrage ausfuehren
    res=self.conn.exec_query(query)
    # Fehler aufgetreten ?
    if res.get_tuple_count()==0:
      # wenn ja, -1 zurueckgeben
      return []
    else:
      #wenn nein Ergebnis zurueckgeben
      dtdnr=res.get_fieldvalue(1,0)

    # Doc-ID's holen
    query="select␣docnr␣from␣doc_description␣where␣dtdnr="
      + str(dtdnr)+";"
    # Abfrage ausfuehren
    res=self.conn.exec_query(query)
    # Fehler aufgetreten ?
    if res.get_tuple_count()==0:
      # wenn ja, -1 zurueckgeben
      return []
    else:
      #wenn nein Ergebnis zurueckgeben
      erg=res.get_resultlist(1,"|",1,0)
      return erg
  #
  # ermittelt die Dokumenten-ID's aller Dokumente, die
  # der durch dtdid gegebenen DTD entsprechen und ein Tag
  # enthalten das searchstr entspricht. Das Ergebnis wird
```

```
# in Form einer Liste zur"uckgegeben.
#
def sgmlsearch_dtdandtags(self,dtdid,searchstr):
  # ermittle Teilergebnisse
  erg1=self.sgmlsearch_tag(searchstr)
  erg2=self.sgmlsearch_dtddocs(dtdid)

  erg=[]
  for i in erg1:
    if i in erg2:
      erg.append(i)

  return erg
```

Klasse c_sgmlstore

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_sgmlstore.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Diese Klasse stellt Methoden bereit, um als
#                    Bitstrom vorliegende SGML-Dokumente in der
#                    Datenbank zu speichern
# Anmerkungen      : Diese Klasse wird von der Klasse c_sgmlread
#                    abgeleitet. Die Varibalen DL_CURRENT_INFILE
#                    und DL_NEW_DOCID werden von der Ablauf-
#                    umgebung gesetzt.
# ---------------------------------------------------------------

# imports
from sc_globals import *
import class_funcs
import pgpy
import c_sgmlread
import urllib
import regsub


class c_sgmlstore(c_sgmlread.c_sgmlread):
      # Variablen
      infile=""
      docid=0
      fd_in=0
      emptytags=[]
      dtdfname=""
      # Variablen fuer Strukturbaumberechnungen
      atag,ntag,atyp,ntyp="","","",""
      anum,nnum=0,0
      stack=[]
      element_counter=0
      ls=("",0)
```

```
#
# Konstruktor
#
def __init__(self,DL_CURRENT_INFILE,DL_NEW_DOCID):
  # uebernehme Parameter
  self.infile=DL_CURRENT_INFILE
  self.docid=DL_NEW_DOCID

  # Konstruktur der Basisklasse
  c_sgmlread.c_sgmlread.__init__(self,self.infile)

  # hole Liste der EMPTY Tags
  self.dtdfname=class_funcs.get_dtdfile(self.get_dtd())
  self.emptytags_lst=self.get_emptytags(self.dtdfname)

  # baue Verbindung zu Postgres auf
  self.conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
  if self.conn.connstatus()!=0:
    print "
      Postgresverbindung konnte nicht aufgebaut werden."
    raise KeyboardInterrupt

#
# oeffnet den eingabebitstrom
#
def open_in(self):
  self.fd_in=open(self.infile,"r")
#
# schliesst den Eingabebitstrom
#
def close_in(self):
  self.fd_in.close()
#
# liest size Bytes vom Eingabebitstrom
#
def read_in(self,size):
  return self.fd_in.read(size)

#
# setzt die aktuelle Position der Lese-/Schreibmarke
# des Eingabebitstroms (Syntax wie Python Befehl seek)
#
#
def seek_in(self,offset,whence):
  self.fd_in.seek(offset,whence)
```

```
#
# liefert die aktuelle Position der Lese-/Schreibmarke
# des Eingabebitstroms zurueck
#
def tell_in(self):
  return self.fd_in.tell()

#
# filtert in der Datenbank abzulegende Text, um sie an
# die PostgreSQL Syntax anzupassen
#
def pgfilter(self,fstr):
  erg=""
  for i in range(0,len(fstr)):
    # ersetze anfuehrungszeichen
    if fstr[i]=="'":
      erg=erg+"\\"

    erg=erg+fstr[i]
  return erg


#
# liefert eine Liste alle in der DTD fname definierten
# EMPTY-Tags zurueck
#
def get_emptytags(self,fname):
  import c_bsio
  import string

  fd=c_bsio.c_bsio(fname,"r")
  erglst=[]
  fd.open()
  buff = "␣"
  while buff!="":
    buff=fd.readline()
    if
       string.upper(string.strip(buff)[:9])=="<!ELEMENT":
      if string.find(buff,"EMPTY")!=-1:
        erglst.append(string.upper(
           string.strip(buff)[10:string.find(
           string.strip(buff),"-")-1]))
  fd.close()
  return erglst
```

```
#
# vermerkt in der Datenbank, dass das Dokument in der
# Datenbank
# gespeichert wurde
#
def register_sgmlstore(self):
  query="
    update␣doc_description␣set␣sgml_stored='t'␣where␣docnr="
  query=query+str(self.docid)+";"
  res=self.conn.exec_query(query)
  if res.commandstatus()[0:6]!="UPDATE":
    print "Fehler␣bei␣Update␣in␣register_sgmlstore"
    return -1
  else:
    return 0

#
# vermerkt die DTD der das Dokument angehoerig ist in
# der Datenbank
#
def save_doctype(self):
  dtdtup=self.get_dtd()
  dtdnum=class_funcs.get_dtdnum(dtdtup)

  query="update␣doc_description␣set␣dtdnr="
  query=query+str(res.get_fieldvalue(1,0))
  query=query+"␣where␣docnr="+str(self.docid)+";"

  res=self.conn.exec_query(query)
  if res.commandstatus[0:6]!="UPDATE":
    print "
      Fehler␣c_sgmlstore.save_doctype␣beim␣Eintragen␣der␣DTDNR"
#
# speichert tagst in der Tag-Klasse der Datenbank
# und berechnet die Strukturbaumverweise automatisch
#
def save_astag(self,tagstr):
  dbclass="doc_tags"
  self.element_counter=self.element_counter+1

  self.nnum=self.element_counter
  self.ntag=tagstr
  isempty=self.is_emptytag(self.ntag)
```

```
# Berechnungen fuer Strukturbaum

if self.ntag[0:2]!="</":
  self.ntyp="S"
  self.stack.append((self.ntag,self.nnum))
else:
  self.ntyp="E"
  self.ls=self.stack[-1]
  self.stack=self.stack[:-1]

if self.ntyp=="S"and self.atyp=="S":
  # atag ist Vater von ntag
  query="update␣doc_tags␣set␣son="+str(self.nnum)
  query=query+"␣where␣docnr="+str(self.docid)
  query=query+"␣and␣elementnr="+str(self.anum)+";"

  res=self.conn.exec_query(query)
  if res.commandstatus()[0:6]!="UPDATE":
   print "FEHLER␣in␣c_sgmlstore.save_astag␣Query␣1"


if self.ntyp=="S"and self.atyp=="E":
  # ntag ist rechter Bruder von anum
  query="update␣doc_tags␣set␣brother="+
   str(self.nnum)
  query=query+"␣where␣docnr="+str(self.docid)
  query=query+"␣and␣elementnr="+str(self.anum)+";"

  res=self.conn.exec_query(query)
  if res.commandstatus()[0:6]!="UPDATE":
    print "
      FEHLER␣in␣c_sgmlstore.save_astag␣Query␣2"

if self.ntyp=="E"and self.atyp=="S":
  # atag ist ein Blatt
  query="update␣doc_tags␣set␣brother="+
   str(self.nnum)
  query=query+"␣where␣docnr="+str(self.docid)
  query=query+"␣and␣elementnr="+str(self.anum)+";"

  res=self.conn.exec_query(query)
  if res.commandstatus()[0:6]!="UPDATE":
    print "
      FEHLER␣in␣c_sgmlstore.save_astag␣Query␣3"
```

```
    if self.ntyp=="E"and self.atyp=="E":
      # atag ist ein Blatt
      query="update doc_tags set brother="+
        str(self.nnum)
      query=query+" where docnr="+str(self.docid)
      query=query+" and elementnr="+str(self.ls[1])+";"

      res=self.conn.exec_query(query)
      if res.commandstatus()[0:6]!="UPDATE":
        print "
          FEHLER in c_sgmlstore.save_astag Query 4"


    # letztes  Tag und Typ merken
    self.atag=self.ntag
    self.anum=self.nnum
    if isempty==0:
      self.atyp=self.ntyp
    elif isempty==1:
      self.atyp="E"
      self.stack=self.stack[:-1]

    # neuen Tagdatensatz in DB eintragen
    tagstr=self.pgfilter(tagstr)
    query="insert into "+dbclass+" values ("+
      str(self.docid)+","
    query=query+str(self.nnum)+",0,0,'"+tagstr+"',"
    query=query+str(0)+");"

    res=self.conn.exec_query(query)
    if res.commandstatus()[0:6]!="INSERT":
      print "FEHLER in c_sgmlstore.save_astag Query 5"

  #
  # speichert textstr in der Text-Klasse der Datenbank
  #
  def save_astext(self,textstr):
    dbclass="doc_text"
    self.element_counter=self.element_counter+1
    textstr=self.pgfilter(textstr)
    query="insert into "+dbclass+" values ("+
      str(self.docid)+","
    query=query+str(self.element_counter)+",0,0,'"+textstr+
      "');"
```

```
  res=self.conn.exec_query(query)
  erg=res.commandstatus()
  if str(erg)[0:6]!="INSERT":
    print "Fehler bei INSERT in Postgres"
    raise KeyboardInterrupt
  return 0
#
# speichert das unter der URL urlstr abrufbare Objekt
# als binaere Daten in der Datenbank
#
def save_asbin(self,urlstr):
  dbclass="doc_bin"
  self.element_counter=self.element_counter+1
  binstr=urllib.urlretrieve(urlstr)
  query="insert into "+dbclass+" values ("+
   str(self.docid)+","
  query=query+str(self.element_counter)+",0,0,'"+
   binstream+"');"

  res=self.conn.exec_query(query)
  erg=res.commandstatus()
  if str(erg)[0:6]!="INSERT":
    print "Fehler bei INSERT in Postgres"
    raise KeyboardInterrupt
  return 0
#
# liest das naechste Tag aus dem Eingabebitstrom und
# gibt es zurueck
#
def get_nextelement(self):
  # leerzeilen ueberspringen
  test1=""
  test1=self.read(1)
  while test1=="\n":
    test1=self.read(1)

  # position des Lese-/Schreibmarke korrigieren
  if test1!="":
    if self.tell()>1:
      self.seek(-1,1)

    # sicherstellen das nicht mitten im Tag gesucht
    # wird
    if test1=="<":
      return self.get_nexttag()
```

```
    else:
      erg=""
      buff=" "
      while buff!="<"and buff!="":
        buff=self.read(1)
        erg=erg+buff

      # letztes Zeichen wegwerfen
      if len(erg)>1:
        erg=erg[0:-1]
      # position des Lese-/Schreibmarke korrigieren
      self.seek(-1,1)
      return erg
  else:
    return ""
#
# ermittelt, ob es sich bei tagstr um ein Tag handelt
# gibt 1 zureuck wenn ja, 0 wenn nein und -1 bei Fehler
#
def is_tag(self,tagstr):
  if len(tagstr)!=0:
    if tagstr[0]=="<"and tagstr[len(tagstr)-1]==">":
      return 1
    else:
      return 0
  else:
    return -1
#
# ermittelt, ob es sich bei tagstr um ein Empty-Tag
# handelt
# gibt 1 zureuck wenn ja, 0 wenn nein und -1 bei Fehler
#
def is_emptytag(self,tagstr):
  import string
  if len(tagstr)!=0:
    if tagstr[1:string.find(tagstr," ")] in
       self.emptytags_lst:
      return 1
    else:
      return 0
  else:
    return -1
```

E.6.4 Paket System (p_system.py)

```
# ------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : p_system.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Paket System, fasst alle Systempaket Klassen
#                    zusammen
#
# Anmerkungen      :
#
# ------------------------------------------------------------

import c_metadoc
import c_metasearch
import c_metaquery
```

Klasse c_metadoc

```
# ------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_metadoc.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Die klasse c_metadoc stellt Methoden zur
#                    Verfuegung um Informationen aus unseren Meta
#                    Docs zu extrahieren
# Anmerkungen      : Diese Klasse wird von der Klasse
#                    c_sgmlread abgeleitet.
# ------------------------------------------------------------

# imports
```

```
import c_sgmlread
import regex
import string
import regsub
from sc_globals import *


class c_metadoc(c_sgmlread.c_sgmlread):
      #
      # Konstruktor, oeffnet das Metadokument
      #
      def __init__(self,fname):
        # Konstruktor der Basisklasse aufrufen
        c_sgmlread.c_sgmlread.__init__(self,fname)
        self.open()

      #
      # liefert eine Liste der Attribute des DOCTYPE
      # Tags zurueck
      #
      def get_dtd(self):
        import string
        self.seek_begin()
        doctypetag=self.get_doctype()
        doctypetag=doctypetag [2:-1]
        return string.split(doctypetag," ")
      #
      # liefert den Inhalt des kompletten
      # HEADER Tags zurueck
      #
      def get_header(self):
        self.seek_begin()
        return self.get_tagcontents("HEADER")
      #
      # liefert den Inhalt des INFO Tags zurueck
      #
      def get_info(self):
        self.seek_begin()
        return self.get_tagcontents("INFO")
      #
      # liefert ein Dictionary mit allen Attributen und ihren
      # Werten des PROPERTY Tags zurueck
      #
      def get_property(self):
        self.seek_begin()
```

```
    erg={}
    buff = self.get_tag("PROPERTY")

    pos1=regex.search("title=*\'",buff)
    pos2=regex.search("author=*\'",buff)
    pos3=regex.search("owner=*\'",buff)
    pos4=regex.search("ident=*\'",buff)
    pos5=regex.search("pubkey=*\'",buff)
    title =string.strip(buff[pos1:pos2])
    author=string.strip(buff[pos2:pos3])
    owner =string.strip(buff[pos3:pos4])
    ident =string.strip(buff[pos4:pos5])
    pubkey=string.strip(buff[pos5:])

    title =title[string.find(title,"\'")+
     1:string.rfind(title,"\'")]
    author=author[string.find(author,"\'")+
     1:string.rfind(author,"\'")]
    owner =owner[string.find(owner,"\'")+
     1:string.rfind(owner,"\'")]
    ident =ident[string.find(ident,"\'")+
     1:string.rfind(ident,"\'")]
    pubkey=pubkey[string.find(pubkey,"\'")+
     1:string.rfind(pubkey,"\'")]

    erg["title"] =title
    erg["author"]=author
    erg["owner"] =owner
    erg["ident"] =ident
    erg["pubkey"]=pubkey
    return erg
  #
  # liefert ein Dictionary alle dem Dokument zugeordneten
  # Methoden und ihrer Namen, sowie MIME-Types zurueck
  #
  def get_methods(self):
    self.seek_begin()
    erg={}
    buff = self.get_tag("METHOD")
    while buff!="":
      pos1=regex.search("name=*\'",buff)
      pos2=regex.search("code=*\'",buff)
      pos3=regex.search("mime=*\'",buff)
      name=string.strip(buff[pos1:pos2])
      code=string.strip(buff[pos2:pos3])
```

```
    mime=string.strip(buff[pos3:])

    name=name[string.find(name,"\'")+
     1:string.rfind(name,"\'")]
    code=code[string.find(code,"\'")+
     1:string.rfind(code,"\'")]
    mime=mime[string.find(mime,"\'")+
     1:string.rfind(mime,"\'")]

    erg[name]=[name,code,mime]
    buff = self.get_tag("METHOD")

  return erg
#
# liefert den Inhalt des Data-Tags zurueck
#
def get_data(self):
  self.seek_begin()
  buff= self.get_tagcontents("DATA")
  return string.strip(buff[string.find(buff,">")+
   1:string.rfind(buff,"</")])

#
# Destruktor, schliesst das Metadokument
#
def __del__(self):
  self.close()
```

Klasse c_metasearch

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_metasearch.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Diese Klasse realisiert ein einfaches
#                    Suchinterface ueber alle gespeicherten
#                    Metadaten
# Anmerkungen      :
# ---------------------------------------------------------------
# imports
import pgpy

class c_metasearch:
      #
      # Konstruktor
      #
      def __init__(self):
        # Verbindungsobjekt zur Datenbank instanziieren
        self.conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)

      #
      # Durchsucht die in der Datenbank gespeicherten
      # Metadaten nach searchstr und betrachtet dabei
      # jeweils nur das durch field gegeben Feld in der
      # doc_description Klasse. Es werden auch aehnliche
      # Ausdruecke gefunden.
      # Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
      # zurueckgegeben
      #
      def metasearch(self,field,searchstr):
        # SQL Abfrage zusammenbauen
        query="select␣*␣from␣doc_description␣where␣"+field+" *"
          +searchstr+";"
        # Abfrage ausfuehren
        res=self.conn.exec_query(query)
        # Fehler aufgetreten ?
        if res.get_tuple_count()==0:
```

```
    # wenn ja, -1 zurueckgeben
    return -1
  else:
    #wenn nein Ergebnis zurueckgeben
    erg=res.get_resultlist(1,"|",1,0)
    return erg
#
# sucht nach ident aehnlichen Dokumenten-ID's
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben
#
def metasearch_ident(self,ident):
  return self.metasearch("ident",searchstr)
#
# sucht nach searchstr aehnlichen INFO-Feldern
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben
#
def metasearch_info(self,searchstr):
  return self.metasearch("info",searchstr)
#
# sucht nach searchstr aehnlichen AUTHOR-Feldern
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben
#
def metasearch_author(self,searchstr):
  return self.metasearch("author",searchstr)
#
# sucht nach searchstr aehnlichen OWNER-Feldern
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben
#
def metasearch_owner(self,searchstr):
  return self.metasearch("owner",searchstr)
#
# sucht nach searchstr aehnlichen TITLE-Feldern
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben
#
def metasearch_title(self,searchstr):
  return self.metasearch("title",searchstr)
```

Klasse c_metaquery

```
# ------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1998
# Dateiname       : c_metaquery.py
# Datum           : 09.12.1997
# letzte Änderung :
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Klasse fuer Abfragen der Metadaten eines
#                   bestimmten Dokuments
# Anmerkungen     :
#
# ------------------------------------------------------------

# imports
from sc_globals import *
import pgpy

class c_metaquery:
      #
      # Konstruktor, setzt Variable
      # ident gibt die Dokuent-ID des Dokuments an
      #
      def __init__(self,ident):
        # Verbidungsobjekt zur Datenbank instanziieren
        self.conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        self.ident=ident

      #
      # liefert den Inahlt des INFO Tags des gespeicherten
      # Dokuments zurueck
      #
      def mquery_info(self):
        query="select info from doc_description where ident="+
          str(self.ident)+";"
        res=conn.exec_query(query)
        return res.get_fieldvalue(0,0)

      #
      # liefert ein Dictionary mit den Attributen und ihren
      # Werten
```

```
# des PROPERTY Tags zurueck
#
def mquery_property(self):
  query="select␣*␣from␣doc_description␣where␣ident="+
   str(self.ident)+";"
  res=conn.exec_query(query)
  erg={'author':'','ident':'','owner':'','pubkey':'',
    'title':''}
  erg['author']=res.get_fieldvalue(0,3)
  erg['ident'] =res.get_fieldvalue(0,5)
  erg['owner'] =res.get_fieldvalue(0,4)
  erg['pubkey']=res.get_fieldvalue(0,6)
  erg['title'] =res.get_fieldvalue(0,2)

  return erg

#
# liefert ein Dictionary mit allen zu den Methoden
# gespeicherten Informationen (MIME-Typ, Name, URL)
# zurueck
#
def mquery_methods(self):
  erg={}

  query="select␣*␣from␣doc_methods␣where␣ident="+
   str(self.ident)+";"
  res=conn.exec_query(query)

  count=res.get_tuple_count()
  j=0
  while j<count:
    mclass=res.get_fieldvalue(j,1)
    mloc =res.get_fieldvalue(j,2)
    mtyp =res.get_fieldvalue(j,3)

    erg[mclass]= [mclass,mloc,mtyp]
    j=j+1

  return erg
```

E.6.5 Paket Maintenance (p_maintenance.py)

```
# ----------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : p_maintenance.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Paket Maintenance, fasst alle Maintenance-Paket
#                    Klassen zusammen
#
# Anmerkungen      :
#
# ----------------------------------------------------------------
import c_maintenance
import c_glimpindex
```

Klasse c_maintenace

```
# ----------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_maintenance.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Diese Klasse realisiert ein paar Methoden
#                    zur Pflege des Systems
# Anmerkungen      :
# ----------------------------------------------------------------

#imports
from sc_globals import *
import pgpy
```

```
class c_maintenance:
      #
      # Reorganisert die Postgresdatenbank
      #
      def vacuum(self):
        self.conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        res=self.conn.exec_query("vacuum;")
        erg=res.commandstatus()
        if str(erg)[0:6]!="VACUUM":
          return -1
        else:
          return 0

      #
      # loescht ein als Bitstrom gespeichertes Dokument
      #
      def delete_bs(self,ident):
        import os
        os.system("rm␣"+"-f␣"+DL_BSOUTPATH+str(ident)+".bs")
        conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        res=self.conn.exec_query(
          "
            update␣doc_description␣set␣bs_stored='f'␣where␣docnr="+
          str(ident)+";")
        erg=res.commandstatus()
        if str(erg)[0:6]!="UPDATE":
          return -1
        else:
          return 0


      #
      # loescht ein Dokument aus dem Speicher
      #
      def delete_doc(self,ident):
        # erstmal als Bitstrom loeschen
        self.delete_bs(ident)
        # dann die Datenbank leeren
        conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        res=self.conn.exec_query(
          "delete␣from␣doc_elements*␣where␣docnr="+
            str(ident)+";")
        res=self.conn.exec_query(
          "delete␣from␣doc_methods␣where␣docnr="+
            str(ident)+";")
```

```
    res=self.conn.exec_query(
      "
        update␣doc_description␣set␣sgml_stored='f'␣where␣docnr="+
      str(ident)+";")
    erg=res.commandstatus()
    if str(erg)[0:6]!="UPDATE":
      return -1
    else:
      return 0
  #
  # fuehrt ein SQL Statement auf der Datenbank aus
  #
  def sql_exec(self,querystr):
    conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
    res=self.conn.exec_query(querystr)
     # Fehler aufgetreten ?
    if res.get_tuple_count()==0:
      # wenn ja, -1 zurueckgeben
      return -1
    else:
      #wenn nein Ergebnis zurueckgeben
      erg=res.get_resultlist(1,"|",1,0)
      return erg

    return erg
  #
  # Reinitialisiert die Datenbank
  # VORSICHT alle gespeicherten Daten gehen verloren
  #
  def dali_reinit(self):
    import time

    print "
      -----------------------------------------------------------"
    print "
      Sie␣sind␣dabei␣die␣komplette␣DALI␣Datenbank␣zu␣loeschen."
    print ""
    print "␣Ich␣warte␣jetzt␣mal␣20␣Sekunden,
      ␣wenn␣Sie␣es␣sich␣anders"
    print ""
    print "ueberlegt␣haben,
      ␣sollten␣sie␣ein␣paarmal␣CTRL-C␣druecken."
    print "
      ------------------------------------------------------------"
    time.sleep(20)
```

```
# Verbindung zu Postgres aufbauen
conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)

# Alle Klassen entfernen
res=self.conn.exec_query("drop table doc_description;"
  )
res=self.conn.exec_query("drop table doc_elements;")
res=self.conn.exec_query("drop table doc_bin;")
res=self.conn.exec_query("drop table doc_tags;")
res=self.conn.exec_query("drop table doc_text;")
res=self.conn.exec_query("drop table doc_methods;")
res=self.conn.exec_query("drop table dtd;")
res=self.conn.exec_query("drop table local_doc_ids;")

# neue leere Klassen erzeugen
res=self.conn.exec_query("
  CREATE TABLE local_doc_ids (docnr int4, dtdnr int4) archive = none;")

res=self.conn.exec_query("
  CREATE TABLE doc_elements (docnr int4, elementnr int4, brother int4,
   son int4, contents text) archive = none;")

res=self.conn.exec_query("CREATE TABLE dtd (dtdnr int4,
   dtdid text,publicname text,
   systemname text) archive = none;")

res=self.conn.exec_query("
  CREATE TABLE doc_tags () inherits ( doc_elements) archive = none;")

res=self.conn.exec_query("
  CREATE TABLE doc_text () inherits ( doc_elements) archive = none;")

res=self.conn.exec_query("
  CREATE TABLE doc_bin () inherits ( doc_elements) archive = none;")

res=self.conn.exec_query("
  CREATE TABLE doc_description (docnr int4, dtdnr int4, title text,
   author text, owner text, ident text, pubkey text,
   info text, metadtdnr int4, bs_stored bool,
   sgml_stored bool) archive = none;")

res=self.conn.exec_query("
  CREATE TABLE doc_methods (docnr int4, meth_class text,
   meth_location text, meth_mimetype text) archive = none;")
```

```
  # Initialen Zaehlerstand einfuegen
  res=self.conn.exec_query("
     INSERT␣INTO␣local_doc_ids␣VALUES␣(100,1);")

  self.read_sgmlcat(self)
#
# traegt alle auf dem System verf"ugbaren␣DTD's␣in␣die
# DTD Klasse der Datenbnak ein
#
def read_sgmlcat(self):
  import c_bsio
  import class_funcs
  import string

  fname=DL_SGMLCATPATH+DL_SGMLCATNAME
  fd=c_bsio.c_bsio(fname,"r")
  fd.open()
  buff="␣"
  while buff!="":
    buff=fd.readline()
    if buff[0:6]=="PUBLIC":
      newdtdid=class_funcs.get_newdtdid()
      publictup=string.split(buff,"\'")

      dtdid=string.strip(publictup[2])
      publicname=string.strip(publictup[1])
      systemname=DL_SGMLCATPATH+dtdid

      query="insert␣into␣dtd␣values␣("+str(newdtdid)+
        ","
      query=query+"'"+dtdid+"',"
      query=query+"'"+publicname+"',"
      query=query+"'"+systemname+"');"

      res=conn.exec_query(query)
      if res.commandstatus()[0:6]!="INSERT":
        print "
           Fehler␣bei␣INSERT␣in␣c_maintenance.read_sgmlcat"

  fd.close()
```

Klasse c_glimpindex

```
# ------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : c_glimpindex.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Klasse für den Zugriff auf glimpseindex
# Anmerkungen      :
# ------------------------------------------------------------

class c_glimp_index:
      #
      # Konstruktor
      #
      def __init__(self):
        from sc_globals import *
        # setzen des Pfades in dem alle als
        # Bitstrom gespeicherten Dokumente liegen
        self.dir=DL_FULLTEXT_DIR
      #
      # Loescht die von glimpse erstellten Indizes
      #
      def remove_index(self):
        import os
        try:
          # loesche Index Files
          os.unlink(self.dir+".glimpse_*")
        except:
          # da waren wohl gar keine da
          pass
      #
      # Erstellt einen neuen Index fuer das
      # Bistromverzeichnis
      #
      def build_index(self):
        import os
        os.system("glimpseindex -f -H "+self.dir+" "+self.dir+
          " >/dev/null")
```

E.6.6 Paket Net (p_net.py)

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : p_net.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Paket Netz, fasst alle Netzpaket-Klassen
#                    zusammen
#
# Anmerkungen      :
#
# -------------------------------------------------------------
import c_tcp
import c_stopwait1
import c_send
import c_retrieve
```

Klasse c_tcp

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : c_tcp.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Eine einfache TCP Connection Klasse
#
# Anmerkungen      :
#
# -------------------------------------------------------------
```

```
# imports
import socket
import time

class c_tcpconn:

      #
      # Konstruktor, wird kein Port mit uebergeben
      # dann wollen wir die Verbindung aufbauen(Client)
      #
      def __init__(self,host="",port=0):
        self.sock=socket.socket(socket.AF_INET,
           socket.SOCK_STREAM)

        if port !=0:
          try:
            self.sock.bind(host,port)
            self.typ="server"
          except socket.error,msg:
            print "---",msg,"---"

        else:
          self.typ="client"


      #
      # baut eine Verbindung mit host,port auf
      #
      def connect(self,host,port):
        self.sock.connect(host,port)

      #
      # wartet auf Verbindungsaufbau
      #
      def accept(self):
        self.sock.listen(5)
        self.peerconn,self.peeraddr = self.sock.accept()

      #
      # schreibt buff in den Socket
      #
      def send(self,buff):
        if self.typ=="client":
          self.sock.send(buff)
```

```
    else:
      self.peerconn.send(buff)

  #
  # wartet auf eine Nachricht die max. buffsize bytes lang
  # ist
  #
  def recv(self,buffsize):
    if self.typ=="client":
      return self.sock.recv(buffsize)
    else:
      return self.peerconn.recv(buffsize)

  #
  # wie recv, wartet aber nur secs Sekunden
  #
  def poll(self,buffsize,secs=1):
    polltime=secs
    erg=""
    if self.typ=="client":
      self.sock.setblocking(0)
    else:
      self.peerconn.setblocking(0)

    while erg==""and polltime>0:
      try:
        if self.typ=="client":
          erg=self.sock.recv(buffsize)
        else:
          erg=self.peerconn.recv(buffsize)
      except socket.error:
        erg=""
        time.sleep(1)
        polltime=polltime-1

    if self.typ=="client":
      self.sock.setblocking(1)
    else:
      self.peerconn.setblocking(1)

    return erg

  #
  # baut eine evtl. bestehende Verbindung ab
  # und schliesst die Sockets
```

```
    #
    def close(self):
      if self.typ=="server":
        self.sock.close()
        self.peerconn.shutdown(1)
      else:
        self.sock.shutdown(1)

      return


    #
    # liefert die lokale Adresse zurueck
    #
    def getaddr(self):
      return [socket.gethostname(),self.sock.getsockname()]

    #
    #liefert die Remote-Adresse zurueck
    #
    def getpeeraddr(self):
      if self.typ=="client":
        return self.sock.getpeername()
      else:
        return self.peerconn.getpeername()
```

Klasse c_stopwait1

```
# ----------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_stopwait1.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Hier ist eine einfache Stop-Wait Protokoll
#                    Maschine als Klasse implementiert.
# Anmerkungen      : Diese Klasse kann verwendet werden, um
#                    aus Dokumentemethoden separate Netzverbindung
#                    zur Datenuebermittlung aufzubauen. Sie ist
#                    von der Klasse c_tcpconn abgeleitet.
# ----------------------------------------------------------------
#

# import
from c_tcp import c_tcpconn
import c_bsio
import select

class c_stopwait1(c_tcpconn):
      #
      # Konstruktor, initialsiert Variable
      # host,port gibt die Adresse des lokalen sockets an
      # buffsize die Paketgroesse bei der Uebertragung
      # timeout die Zeit (in Sekunden) die auf eine Bestaetigung
      # gewartet wird, mittels debug=1 werden zusaetzliche
      # Nachrichten  ueber den Verlauf der uebertragung
      # ausgegeben
      #
      def __init__(self,host="localhost",port=0,buffsize=512,
        timeout=5,debug=0):
        # Konstruktor der Basisklasse aufrufen
        c_tcpconn.__init__(self,host,port)
        self.blocksize=buffsize
        self.timeout=timeout
        self.debug=debug
      #
      # Destruktor, trennt die Verbidung falls sie aufgebaut ist
```

```
  #
  def __del__(self):
    try:
      c_tcpconn.sock.close()
      c_tcpconn.peerconn.close()
    except:
      pass
  #
  # fuehrt einen TCP connect an  die durch
  # host,port gegebene Adresse aus
  #
  def connect(self,host,port):
    c_tcpconn.connect(self,host,port)
  #
  # wartet auf Verbidungsaufnahme durch eine Gegenstelle
  #
  def accept(self):
    c_tcpconn.accept(self)

  #
  # sendet sendstr mittels des Stop-Wait Protokolls
  # dabei wird die Nachricht segmentiert (blocksize)
  # und die Uebertragung eines Pakets bei
  # Uebertragunsfehlern
  # bis zu 10 mal wiederholt
  #
  def send_str(self,sendstr):
    hilf=sendstr
    errcount=0
    muell=self.poll(self.blocksize+1,1)

    while len(hilf)>0 and errcount < 10:
      self.send(hilf[:self.blocksize])

      if self.debug:
        print "send:",hilf[:self.blocksize],
           len(hilf[:self.blocksize])

      ack=self.poll(8,self.timeout)

      if self.debug:
        print "recv:",ack

      if ack=="ACK"+str(len(hilf[:self.blocksize])):
        hilf=hilf[self.blocksize:]
```

```
    else:
      errcount=errcount+1

  # Konnten wir alles uebertragen
  if errcount<10:
    ackend=0
    while ackend==0 and errcount <10:
      self.send("END")
      ack=self.poll(8,self.timeout)

      if self.debug:
        print "recv:",ack

      if ack=="ACK3":
        ackend=1
      else:
        errcount=errcount+1
  # Rueckgabewert ermitteln
  if errcount < 10:
    return 0
  else:
    return -1

#
# wie send_str, nur wird die Datei fname gesendet
#
def send_file(self,fname):
  fd=c_bsio.c_bsio(fname,"r")
  errcount=0
  buff=" "
  fd.open()
  while buff!=""and errcount < 10:
    buff=fd.read(self.blocksize)
    self.send(self,buff)
    ack=self.poll(8,self.timeout)
    if self.debug:
      print "recv:",ack
    if ack!="ACK"+str(len(buff)):
      fd.seek(-len(buff),0)
          errcount=errcount+1
  # Konnten wir alles uebertragen
  if errcount<10:
    ackend=0
    while ackend==0 and errcount <10:
      self.send(self,"END")
```

```
        ack=self.poll(8,self.timeout)
        if self.debug:
          print "recv:",ack
        if ack=="ACK3":
          ackend=1
        else:
          errcount=errcount+1
    fd.close()
    # Rueckgabewert ermitteln
    if errcount < 10:
      return 0
    else:
      return -1


  #
  # empf"angt␣eine␣Zeichenkette␣von␣␣der␣Gegenstelle
  #
  def recv_str(self):
    erg=""
    buff=""
    if self.typ=="server":
      select.select([self.peerconn.fileno()],[],[])
    else:
      select.select([self.sock.fileno()],[],[])

    while buff !="END":
      buff=self.poll(self.blocksize,self.timeout)

      if self.debug:
        print "recv:",buff

      if buff !="":
        if buff !="END":
          erg=erg+buff
        self.send("ACK"+str(len(buff)))

        if self.debug:
          print "send:ACK"+str(len(buff))

      else:
        self.send("ERR")

        if self.debug:
          print "send:ERR"
```

```
  return erg
#
# schreibt alle empfangenen Daten in die Datei mit
# dem Namen fname
#
def recv_file(self,fname):
  fd=c_bsio.c_bsio(fname,"a+")
  fd.open()
  buff=""
  while buff !="END":
    buff=self.poll(self.blocksize,self.timeout)
    if self.debug:
      print "recv:",buff
    if buff !="":
      if buff !="END":
        fd.write(buff)
      self.send("ACK"+str(len(buff)))
      if self.debug:
        print "send:ACK",str(len(buff))
    else:
      self.send("ERR")
      if self.debug:
        print "send:ERR"
    fd.close()
    return 0
```

Klasse c_send

```
# --------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_send.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Diese Klasse versendet Teile eines bestimmten
#                    Dokuments ueber ein separate Netzverbindung
# Anmerkungen      : Diese Klasse wird von den Klassen c_sgmlquery
#                    und c_stopwait1 abgeleitet.
#
# --------------------------------------------------------------
#
class c_send(c_sgmlquery,c_stopwait1):
      #
      # Konstruktor
      # ident gibt die Dokument-ID an
      # peerhost, peeport geben die Netzadresse des Peers an
      # an diese Adresse werden alle Daten gesendet
      # blocksize gibt die Paketgroese an (s. c_stopwait1)
      # timeout gibt den timeout fuer das Warten auf Paket-
      # bestaetigungen an.
      #
      def __init__(self,ident,peerhost,peerport,blocksize,
        timeout):
        # Konstruktoren der Basisklassen aufrufen
        c_sgmlquery.__init__(self,ident)
        c_stopwait1.__init__("",0,blocksize,timeout)
        self.peerport=peerport
        self.peerhost=peerhost
        self.connect(self.peerhost,self.peerport)
        self.ident=ident
      #
      # Destruktor
      #
      def __del__(self):
        # Destruktor der stopwait Klasse aufrufen
        c_stopwait1.__del__()
      #
```

```
# sendet als Bitstrom gespeicherte Daten
# und zwar von der Position start bis Position end
# wobei start und end absolute Positionen innerhalb
# des Bitstroms bezeichnen
#
def send_bs(self,start,end):
  import c_bsio
  from sc_globals import *
  fname=DL_BSOUTPATH+str(ident)+'.bs'
  fd=c_bsio.c_bsio(fname,'r')
  fd.seek(start)
  while fd.tell()<=end:
    buff=fd.readline()
    self.send_str(buff)

  fd.close()
  del fd
#
# sendet den Inhalt des tagcounten Tags mit Namen
# tagname an den Peer
#
def send_tagcontents(self,tagname,tagcount):
  self.send_str(self.sgmlquery_tagcontens(tagname,
     tagcount))
  return 0
#
# sendet den das tagcounte Tag mit Namen
# tagname an den Peer
#
def send_tag(self,tagname,tagcount):
  self.send_str(self.sgmlquery_tag(tagname,tagcount))
  return 0
#
# sendet die Tag-Struktur (Strukturbaum) des Dokuments
# an den Peer
#
def send_structure(self):
  self.send_str( self.sgmlquery_structure())
  return 0
#
# sendet das gesamte Dokument an den Peer
#
def send_doc(self):
  self.send_str(self.sgmlquery_doc())
  return 0
```