TYPES 1.0

Reusable objects construction support

Author: Diego Ernesto Malpica Chauvet

INTRODUCTION

The goal of Types is foment the creation of reusable software components, giving an adequate support for the creation of generic classes. This is obtained increasing the RUNTIME support for the data types.

Types works with handlers that extends capacities of the data types. The handlers could be created for any data type in a really easy way.
This handlers provide general algorithms that can be especialized in orther to obtain a better preformance for a especifisc tasks.

A handler extends the data types with:

- A garbage collector.
- Support for dynamic data structures.
- Support for recursive data structures..
- Support for multiple referenced data.
- Local and External data representation encode and decode.
- code and decode to and from files.
- Support for code and decode of polymorfic objects.
- Extensive use of dynamic memory.
- Support for types created using handlers.
- Support for opaque data types (types no created necessarily using handlers).
- named data types.
- Use of Exception for error reporting.

This characteristics are exploded and powered by several interfaces designed to access them.

To name classes a _C postfix is used when the class is not a handler. When a class is a handler a _B postfix is used (for example: stack_B).

See the example provide with this text for a general look up.

LEGAL INFORMATION

FREE EDUCATIONAL AND EVALUATION USE

Subject to the provisions contained herein, the Author Diego Ernesto Malpica Chauvet hereby grants you a non-exclusive license to use its accompanying proprietary software product ("Software") free of charge if (a) you are a student, faculty member or staff member of an educational institution (K-12, junior college or college) or an employee of a non-profit organization; or (b) your use of the Software is for the purpose of evaluating whether to purchase an ongoing license to the Software. If you do not fit within the description above, a license fee is due to the Author and no license is granted herein. If you are using a free version of the Software, you will not be entitled to support.

Title, ownership rights, and intellectual property rights in and to the Software and Documentation shall remain in the Author.

This Agreement does not include the right to sublicense the Software and is personal to you and therefore may not be assigned (by operation of law or otherwise) or transferred without the prior written consent of the Author.

Either party may terminate this Agreement immediately in the event of default by the other party. Upon any termination of this Agreement, you shall immediately discontinue the use of the Software. Your obligations to pay accrued charges and fees shall survive any termination of this Agreement.

REGISTRATION (DEVELOPER LICENSE)

The registration fee is $29 U.S. dollars (minimum).
You will get a registration ID number and a Developer license.

The Developer license let you use Types software in your products, mean while no source code of Types software is included. If you need to supply source code of Types software with your products the end user must obtaine his Types License.

Title, ownership rights, and intellectual property rights in and to the Software and Documentation shall remain in the Author.

The Developer license does not include the right to sublicense the Software and is personal to you and therefore may not be assigned (by operation of law or otherwise) or transferred without the prior written consent of the Author.

Either party may terminate the Agreement immediately in the event of default by the other party. Upon any termination of this Agreement, you shall immediately discontinue the use of the Software. Your obligations to pay accrued charges and fees shall survive any termination of this Agreement
.
For registration Send the registration form in the next page along with your payment in US currency to: DIEGO ERNESTO MALPICA CHAUVET, Verdin No. 2 , Atizapan de Zaragoza, CP 54500, Edo. Mexico, Mexico. You can also register thruw compuserv: Program Title: TYPES 1.0 Registration ID: 5211

If you have any questions, comments or suggestions please contact Diego Malpica at the above address or e-mail malpica@mailer.main.conacyt.mx. Site license prices are given on request.

As this software is shareware it comes 'as is', there is no support provided. However, if you discover bugs or problems please contact the developer at the above e-mail address.

REGISTRATION FORM TYPES 1.0

DIEGO ERNESTO MALPICA CHAUVET
Verdin No. 2
Atizapan de Zaragoza
CP 54500
Edo. Mexico
Mexico

NAME: ____
COMPANY NAME: ____
ADDRESS: ____
____
____
____
____
PHONE NUMBER: ____
E-MAIL ADDRESS: ____
(Internet Only)

GENERAL DESCRIPTION

The handlers can be classified en three main groups:

1. basic data types handlers (opaque).
2. handlers for types created using handlers.
3. named data types handlers (generally used for polymorfic objects).

The basic data types handlers are the ones that could be constructed from a type that is not necessarily implemented using handlers. In this case it is necessary to include an specialization for the de coding routine. Types already includes handlers for the following basic types: char_B, double_B, float_B, int_B, long_B, short_B, u_char_B, u_int_B, u_long_B and string_B.

Types also includes several complex data types, includes a double linked list (dll_C), a stack implemented using the double linked list (stack_C), and storage class(with transactions support) for variable size objects (storage_C), and a btree with transactions support(btree_C).

A handler for this complex data types could be made like this:

```
bt_C< dll_C < bt_C < stack_C < int_B > > > >   list;
```

In this example list is a handler to a double linked list of handlers to stacks to handlers to ints.
You can also say a double linked list of stacks of ints.

You can define new types to symplify the work:

```
bt_C< stack_C<int_B> >   stack;
```

```
typedef bt_C< stack_C<int_B> > stack_B;
stack_B stack;
```

This two ways are equivalents, in both ways you have a stack of ints.

Handlers Characteristics

The handlers have a set of characteristics that are used to manipulate the way they hand the objects.

name_BT
- is a macro used to set the name.
- if used must be unique.
- must be used with polymorfic objects.

BT_BA
- is a symbolic constant flag.
- is used to indicate that the handler is a handler of a opaque data type.
- since the type is opaque you must suply a especial coding metod to the handler.
- a handler must be BT_BA or BT_BT, it can´t be both.

BT_BT
- is a symbolic constant flag.
- is used to indicate that the handler is a handler of a type implemented using handlers.
- a handler must be BT_BA or BT_BT, it can't be both.

ECODE_F
- is a symbolic constant flag.
- this flag is used when you impliment a especial code metod for a handler this flag indicates use especial code metod.
- is used in conjuntion with BT_BA

ECODEA_F
- is a symbolic constant flag.
- this flag is used when you impliment a especial code metod for a handler this flag indicades use the especial code metod after the normal coding escheme.
- is used in conjution with BT_BT to perform adicional coding operations.

BT_CA
- is a symbolic constant flag.
- if this flag is not set in the handler characteristisc the handler is skipped in the coding operations encode,decode,ensize,clone,dup....

ecode
- is a macro used to create the coding especialized metod.

- it must be unique since it is a definition.

The handlers are by default: BT_BT, BT_CA and not named.

Changing handlers characteristcs

BT_BA, BT_BT, and BT_CA can be set in three diferents forms.
1. as type characterists.
2. as creation characterists.
3. as dynamic characterists.

Who do they work together?
If you do not specify type characteristics the handler type characteristics will be by default BT_BT and BT_CA. If you do not specify creation characteris when you create the handler the creation characteristics will be set to be the same as the type characterics. When you create a handler the dynamic characsteristics are set to be the same as the creation characteristics.
You can modify de dynamic characteristics at any time in the life of the handler.

Type characteristics

You define the type characteristics as a especialitacion of the handler. You must use the **bt_BT** macro, this macro takes two parameters the first is the handler type and the secondo parameter especifies the characteristics to be set. This macro is a definition and should apeare only once in your code, for this reason you can´t put it in a header files with the declarations that are going to be included in various modules.
Examples:

```
    typedef bt_C<int>  int_B; //type declaration
    bt_BT(int_B,BT_BA|BT_CA);  //type characteristics definition

bt_BT(bt_C<char>,BT_BA|BT_CA); //type characteristics definition
```

Creation characteristics

You define this characteristics as a parameter for the constructors of the handler.
Examples:

```
    int_B       var(BT_BA);
```

Dynamic characteristics

You define this characteristics whit the following metods:

```
    void put_bt(char bt);    //clear all the characteristics of the handler then set the this characteristics to the indicated by bt
    void set_bt(char bt);    //set the characteristics of the handler bt conteins the characteristcs to be set
    void reset_bt();         //change the handler characterists to the creation characterists
    void clear_bt(char param);          //clear the indicated characteristics of the handler
```

Example:

```
    int_B var(BT_BA|BT_CA);//creation characteristics
    var.clear(BT_CA);//dynamic characteristics
```

name_BT is a macro that is used in way like the bt_BT macro, it must be unique since it is a definition. This macro uses two parameter the first parameter indicates the handler, the second parameter is a string constant that must identify the handler type in an unique way.
Examples:

```
    typedef bt_C<int>  int_B; //type declaration
    bt_BT(int_B,BT_BA|BT_CA);  //type characteristics definition
    name_BT(int_B,"integer_B");

bt_BT(bt_C<char>,BT_BA|BT_CA); //type characteristics definition
    name_BT(bt_C<char>, "char_B");
```

ecode is macro that is used to create a especial code metod and to select the kind of especial code ECODE_F or ECODEA_F, you can´t use both. The especial code metod recieve three parameters (void* a,int n_elements,gencode_C& gencode). The parameter a is a pointer to an array of objetcs, n_elements is the number of elements of the array and gencode is a coding control class that support coding, decoding and sizing in Local data representation or External data representation to an from files or dumps of memory. This class is pased to the metod totaly inicialized, and with the pointers set to perform the next operation.
The next example shows the especial code metod used for handler of the opaque data types char of th handler char_B.
The macro uses two parameters the first indicate the handler and the second the kind of especial code.

Example:

```
ecode(char_B,ECODE_F)
{
        gencode.dr_code(*(char*)a,n_elements);
         return 1;
}
```

The next examples show handlers with various characteristics::

```
typedef bt_C < dll_C <int_B> >  dll_int_B;
// BT_BT and BT_CA  handler of a list of ints.

typedef bt_C<char> char_B;
bt_BT(char_B,BT_BA|BT_CA);
ecode(char_B,ECODE_F)
{
        gencode.dr_code(*(char*)a,n_elements);
        return 1;
}

// this is a handler for a opaque types
// BT_BA, BT_CA, ECODE_F;
//note that a especial code routine is provided.
```

The headers files document the methods and data members of the different classes , extracts of the headers files are included in apendix A.
You can saw its functionality in the examples provided.

Garbage Collector

Who it works?

Every object (array objects) has a reference counter, when the reference count reach zero the object is deleted. This is not the only case when a object is deleted, a object can be deleted if all the references to it are recursive that means that it have not external references.
External rereferences are the references provided by a handler that can not be accesed throw the object.
When a handler is delete and the number of references to the object it references if any, the garbage collector launch´s a algorythm to determine the number of external references, if the number of externals references reachs 0 the object is delete.
The analisis of external refences can consume many resources of your systems. So you can overide the standard algorythm by creating a secure reference, the tecnique to do this is to create a handler with the especific task to be a external refence, once you have make the handler you must reference the object to be secure and then secure the reference. You can unsecure the object with the release method. The secure method makes the handler a secure reference and increments the secure reference count in the object. The release method makes the handler unsecure and decrements the secure reference of the object.
When a handler is deleted, the references to the object are decreased and if the handler was secure then the handler is release (unsecured) and the secure references count in the object is decreased.
When a handelr is deleted it can determine if after the handler is deleted the object will still have a secure reference, if these is true the object will not be deleted.
You dont have to delete a handler to start the garbage collector. When a handler is deleted its contructor calls the forget method, you can call the forget method any time. Also if you make a handle to refence the object referenced by another handler, the actual reference is forgetten.

Polymorfic Coding and Decoding

You must use named handlers in order to use polimorfic Coding and Decoding. You also must register your type in a data base. If you have not register your type in the data base a exception will be thrown when you try to decode a object of a type that is not registred.

How do I register my types?
The types are automatic registered when you create the first instance of a handler.

There is a global variable that determine the default coding method the variable name is DEF_CODE you can set this variabler to LDR_CODE or XDR_CODE, note that a XDR lybrary must be used in order to use XDR_CODE

The next example helps to illustrate how Types works

```
#include"stack.h"
#include<iostream.h>


class person_C {
   string_B name;
   int_B age;
   person_C() {name.create(); age.create(); age()=0;}
};


typedef   bt_C<person_C>  person_B;


typedef   bt_C< stack_C<person_B> >  stack_person_B;

int main()
{

    stack_person_B  stack;
    bt_C<stack_C < bt_C <person_C> > > > stack2;
    person_B   person;

   stack.create();
   person.create();
   person().name()="Diego Ernesto";
   person().age()=25;

  stack().push(person);

  stack2.clone(stack);
  stack2().top()().name()="Emilio";
  stack2().push(person);

  cout<<(char*)stack2().top()().name()<<"\n";
  stack2().pop();
  cout<<(char*)stack2().top()().name()<<"\n";
  cout<<(char*)stack().top()().name()<<"\n";
  return 0;

}
```

Example description:

- The first line includes de header file for a stack.
- The second line includes the standard header file iostream.h
- A new class person_C is made from to handlers the first handler is for a string_C type and is named name, the second handler is a handler for a int type.
- The constructor of the class person_C does: creates an instance of the class string_C, creates an instance of the int basic data types. Acces the instance previusly created by the age handler and assigns 0 to it;
- Use typedef to create an alternate name for new data handlers
- A data handler for the person_C class.
- A data handler for stack_C of data handlers to person_C
- in the body of main a handler to stack_C instanced and the name of the instance is stack.
- unsing another notation a new instance of the same previus type is created
- an instance of the stack is created.
- an instance of a person is creater.
- a string is asigned to the name element of the instance of person.
- an int is asigned to the age elemente of the instance of person.
- the person is pushed into the stack.
- an indepent copy of the stack is created in the stack2 handler, this is a replica, a clone of the stack instance handled by stack.
- the stack instance in the handler stack2 is acceced, the instance in the top of the stack is acceced and the element name of this istance is acceced to assign a new string to it.
- the instance of a person handled by de handler person is pushed in the stack handled by the stack2 handler.
- display the name of the person at the top of the stack
- make a pop to the stack
- display the name of the person at the top of the stack
- display the name of the person at the top of the stack
- return 0

Compiling

The basics files are btypes.cpp, types.cpp, gencode.cpp, btypes.h types.h, gencode.h, and dll_bas.h. Several others files are included testX.cpp, dll.h, stack.h, dllk.h, storage.h, action.cpp,storage.cpp, btree.h, btree.cpp,key.h.key.cpp that are used in various examples.

Types is based in several characteristics of C++ that will be in the C++ standard, so you will need a compiler that implement those. These are templates, and exceptions. I have fully test Types with Borland C++ 4.0 in DOS and Windows. If you have problems first check that you have the latest version of you C++ compiler. When using Borland C++ disable precompiled headers options.

Types can use a XDR (External Data Representation) library. these has to be an ONC (Open Network Computing) implementation like the one of SUN. If you haven't a library like this you can disable this feature by commenting the witch define the XDR_LIBRARY symbol in the file gencode.cpp. If you enable this feature you must disable the XDR_LIBRARY_BCPP this simbol is also in the gencode.cpp file.

Types can full work with out a XDR library if you are using Borland C++ 4.0, in order to do this you must make sure that the XDR_LIBRARY symbol is not defined and the XDR_LIBRARY_BCPP is defined, this are the default setings.

If you are not using Borland C++ 4.0 and you havent a XDR library you most make sure that the XDR_LIBRARY and XDR_LIBRARY_BCPP simbols are undifined. By this way you can not use external data representation in your program, if you tray tou use it an exception will be thrown.

There are several simbolic constans that you can use for debuging. In varios tests the DEBUG_BT symbolic constant is defined, so the number of objects referenced by the handlers is displaayed when a object is created or deleted.

There are various *.ide files that you can use to build the test examples using Borland C++ 4.0 if you are not Using this IDE files remember to define the symbolic constant DEBUG_BT if you want to watch when a object is created and when is deleted.

Watch for compilations obtions the pointer to the virtual pointer table in virtual objects must be the first data member of virtual clases, this is default in varios compilers.

Note that especialization for methods of generic clases are used, so you may need to put this especialization in a table, that is the case of SPARCcompiler C++ of SUN.

Types is formed by this programs

(clases)

1. bt_C<T>
2. dllk_C<T>
3. dll_BAS<T>
4. dll_C<T>
5. dll_node_C<T>
6. dln_BAS<T>
7. stack_C<T>
8. dllk_C<T>
9. btree_C
10. btree_item_C
11. btree_page_C
12. extra_C
13. gencode_C
14. info_C
15. key_C
16. storage_C
17. storage_control_C
18. storage_page_C
19. storage_trans_control_C
20. string_C
21. types_C
22. type_info
23. type_C
24. extra_C
25. gen_C
26. direccion_C
27. datos_personales_C
28. action_C
29. except_C

Notes

If you create libraries using Types, that you think that can be general interest, and you want to make then freeware please send me a copy by Email and I will try to distribute them with the types package.

APENDIX A
(Headers Files Stracts)

```
=======================================================
              types.h documentation
=======================================================
//
//(C) Copyright 1995, Diego Ernesto Malpica Chauvet.
//           All rights reserved.

#include <stdlib.h>
#include "except.h"
#include "gencode.h"
#include "dllbas.h"

//General simbolic constants
#define USE       1
#define CREATE    2
#define XDR_CODE    1    //external data representation
#define LDR_CODE  2    //local data representation
extern char DEF_CODE;    //Default code metod (LDR)
                    //you can change it at run time
                    //or before compiling it definition is in file
                    //types.cpp use XDR_CODE or LDR_CODE to modify it.
#define ENCODE    1    //used to encode
#define DECODE    2    //used to decode
#define ENSIZE    3    //used to calculate the size of the code

//Characteristics of the handler (bt, init_bt bt_s() , set_bt()...)
#define BT_CA   (char)1    //BT  CARE  if this flag is not set
                     //the handler will be skipped when coding sizing
                     //and decoding are doing.
#define BT_BT   (char)2    //BT  BT the handler is of a type implemented with
                     //handlers and general algorithms should be used
#define BT_BA   (char)4    //BT  the handler is of an opaque type a especial
                     //coding algorithm should be provided.
//Coding characteristics of the handler
#define ECODE_F  (char)2    //Especial coding tag
#define ECODEA_F (char)4    //Normal coding and especial coding  after normal
                      //conding


void   chknew(void* param);//funtion used to check memory allocation

//class type is used to save type dependant data and metods so it can be used
//by general pointers.
class type_C {
public:
//------------------type dependent data--------------------------
  char  flags;         //BT characteristics of the type
  char* name;          //name of the type
//------------------type dependent metods------------------------
  void* (*s_new)              (int n);
  void  (*s_delete)           (void * a);
  int   (*s_lsize)            ();
  int   (*s_ecode) (void* a,int n_elements,gencode_C& gencode);
//--------------------------------------------------------------
type_C();
};
class info_C {
public:
  void      *extra;  //extra data member (internal use)
  int       references;//references counter
  int       elements;  //numbet of elements in the array of objects
  int       secure;    //secure counter 0 if is not secured
  void      *ptr;      //pointer to the array of objetcs
  info_C();
};
class types_C {
public:
```

```
  static  long   int n_bt;  //number of arrays of objects referenced
                       //by all the handlers
  type_C*       type;      //pointer to type (data type information).
  info_C*       info;      //object control data and
                       // indirect pointer to array of objects.
  char          bt;        //handler characteristics.
                       //can be modified any time.
                       //by default take the creaation characteristics
  char          init_bt;   //handler creation characteristics.
                       //this are specified by a contructor.
                       //by default take the type characteristics
  char          secure_f;  //secure reference flag(internal).
  int  is_void();          //return true if handler reference is invalid.
  int  elements();         //number of elements of the array of objects
                       //referenced by the handler
  int  redim(int n);       //change de size of array of objects to n elements
  void put_bt(char bt); //clear all the characteristics of the handler then
                    //set the this characteristics to the indicated by
                    //bt
  void set_bt(char bt);//set the characteristics of the handler
                    //bt conteins the characteristcs to be set
  void reset_bt();         //change the handler characterists to the creation
                       //characterists
  void clear_bt(char param);//clear the characteristics indicated
                       //of the handler
  void secure();           //make a secure reference
  void release();          //unmake a secure reference
  int  size(int op=DEF_CODE);//size in bytes of the genetic code of the objects
  static  int  s_code_real        (void *a,gencode_C& gencode);
  //this function recieves a pointer to a handler in the param a,
  //the gencode param must be previous initialised an ready to work
  //if you are creating a especial code metod you can use the gencode param
  //pased to the especial code metod this is ready to work with it.
  //The s_code_real metod provide access to the general coding algoritm.
  //This metod does not prefix the lenght of the gen. code in the gen. code.
  int  encode(void* dump ,int op=DEF_CODE);//generete the genetic code
                                  //of the objects
  int  decode(void* dump ,int op=DEF_CODE);//generete the objects
                                  //from the genetic code.
  int  encode(FILE*file ,int op=DEF_CODE);//generete the genetic code
                                  //of the objects
  int  decode(FILE*file ,int op=DEF_CODE);//generete the objects
                                  //from the genetic code.
      //this methods always use the size of the code as a prefix in the code
      //this size type is int.
      //op parameter can take three diferents values
      //DEF_CODE use the default representation for the genetic code.
      //XDR_CODE use external data representation for the genetic code.
      //LDR_CODE use local data representation for the genetic code.

      //dump parameter is a pointer to a char_B objetc, when encoding if
      // the number of elements if less than is needed the char_B object
      // is redimed, if the reference is void the char_B object is created.

      //file paremeter is a pointer to an open file, make sure that this
      //file is open for writing if you are going to code into it.

};
template<class T>
class bt_C:public types_C {
public:
  static type_C* s_type; //pointer to the type information
                    //and metods that depend of the type.
  static char&  bt_s();  //characteristics of the handler type
  static char*& name_s();//name of the handler type
  int   create(int n=1);  //create an array of n objects
  void  secure();         //make this a secure reference
  void  secure(bt_C<T>& op);//reference the objects referenced by op
                       //and secure this new reference.
  int   clone(bt_C<T>& a);//make a clone of the objects referenced by
```

```
                    //and reference it, if the handler previously had
                    //a valid reference this previus reference is
                    //forgeten.
  bt_C<T> dup();//clone itself and return a new handler that references the
           //objects cloned
  int   equal(bt_C<T>& a); //compare the info contained by objects
                    //resursivily.
  bt_C<T>& operator=(bt_C<T>& op); // reference the objecst of the handler op
                         // and forget the previous reference.
  bt_C<T>& operator=(void* op);//same ass the previous metod but in this case
    //a pointer to a handler is recieved.
    //Yo can sen pointers to bt_C<S> where bt_C<S>  where class S is a class
    //sun of T. T and S must be named data types, in order to use this metod
  int operator==(bt_C<T>& op);
    //compare only the references
  int operator==(void* op);
    //compare only the references op is a pointer to a handler
  T& operator()();
    //access the first element of the array of objects.
  T& operator()(int n);
    //access the n element of the array of objects.
    //n=0 for the first element
  void forget();
    //forget the reference and make this reference void.
    //if the handler is secure and is not the last reference to the object
    //then the forget will not analize if the objetc should be destroyed
    //if the object is secure you indicate that at least you have one more
    //external referece(may be indirect) to the object.
  bt_C(char bt=0);
    //handler construtor bt are the handler construction characteristics
  bt_C(bt_C<T>& op,char bt=0);
    //handler constructor bt are the handler construction characteristics
    //the objects referenced by op will be referenced by this new handler
 ~bt_C();
    //handler destructor.
    //a forget is made before the handler is destroyed.
};

//general handlers and general handler pointer types
class gen_C {};
typedef bt_C<gen_C> gen_B;
typedef gen_B*      gen_P;


//especializations
#define bt_BT(a,b) char& a::bt_s(){static char var=(b);return var;}
//this macro is used to provide a especial BT
#define name_BT(a,b) char*& a::name_s(){static char* var=(b);return var;}
//this macro is used to name the class
#define ecode(b,c)  int b::ecode_type(){return c;}; \
               int b::s_ecode(void* a,int n_elements,gencode_C& gencode)
//the function declared whit this macro would recieve three params
//a  pointer to the data
//n_elements the number of elements referenced by the pointer
//gencode a reference to gencode_C class that most be used for the coding
//       operations

========================================================
========================================================
```

```
=======================================================
                btypes.h documentation
=======================================================
//
//(C) Copyright 1995, Diego Ernesto Malpica Chauvet.
//            All rights reserved.
//          basics types header file

#include "types.h"

//basic types handlers types declarations
typedef bt_C<char> char_B;
typedef bt_C<double> double_B;
typedef bt_C<float> float_B;
typedef bt_C<int> int_B;
typedef bt_C<long> long_B;
typedef bt_C<short> short_B;
typedef bt_C<unsigned char> u_char_B;
typedef bt_C<unsigned int> u_int_B;
typedef bt_C<unsigned long> u_long_B;

class string_C:public char_B
{
  public:
  string_C& operator=(char* str);//copy string
  string_C& operator+=(char* str);//concatenates strings
  int      compare(char* str); // string compares
  operator  char  * (); //return a pointer to the string
};

typedef  bt_C<string_C>  string_B;// type dedclaration for a handler

=======================================================
=======================================================
```

```
========================================================
                  action.h documentation
========================================================
//This class is the basic framework to implement actions

class action_C
{
public:
   virtual int start();
   virtual int finish();
   virtual int doaction(void* a=NULL);
};

typedef bt_C<action_C> action_B;
========================================================
========================================================
```

```
=====================================================
            gencode.h documentation
=====================================================
//
//(C) Copyright 1995, Diego Ernesto Malpica Chauvet.
//          All rights reserved.

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

// the next two simbols can be defined in the gencode.cpp file
//#define XDR_LIBRARY //This symbol should be defined if you are planing to use
            //and external data representation library SUN XDR.
            //the simbol xdr_LIBRARY_BCPP should be undefined if
            //this simbol is defined
//#define XDR_LIBRARY_BCPP  //This simbol should be defined if you haven't
               //a XDR libary and you are using borland C++
               //or if you want to use the you the XDR provided
               //with types for borland C++ 4.0.
               //You must undefine XDR_LIBRARY
               //if you use this symbol.

//gencode_C encapsulates a set of routines used to encode, decode and size
//basic data types like int char long int float double, ...
//you can use this class in two ways you can use a file to the
//operations or you can use a memory dump.
//In order to use a memory dump you have to use the begin metod that have
//the dump_base_param, this is the base of the memory dump tha will be used
//if you want to use a file you must use the metod that have the file_param
//parameter.
//You can avoid using this begin metods but you have to supply at least the
//same funcionality ther provide.
//
//The parameter operation_param must be ENCODE , DECODE  or ENSIZE
//   that are enums defined in signal.h
//The parameter DR_param  must be LDR_CODE or XDR_CODE
//   that are enums defined in signal.h
//When you finish your encoding operations you must call the metod end


//For files and memory dumps:
//when you encode or decode data the pointer is moved.
//when you ensize data de pointer is not moved
//the base pointer is never moded


class gencode_C {
public:
   char* dump_base;  //Base of the dump of memory used in operations
   char* dump_pt;    //pointer mark the position of the next operation in the
              //memory dump
   FILE* file;       //pointer to the file
   unsigned long file_base; //pointer to the base position in the file
   unsigned long file_pt;   //pointer that marks the position of the next
                 //operation in the file
   int   operation;  //type of operation to be performed ENCODE DECODE ENSIXE
   int   size;       //size in bytes of the operation bytes
   int   DR;         //tyde of data representation to by used LD_CODE XDR_CODE
   void * lista;     //extra data for internal use. dont'modify it.
   gencode_C();

virtual   int begin(char* dump_base_param,int operation_param,int DR_param);
virtual  int begin(FILE* file_param,int operation_param,int DR_param);
virtual   int end();
//Code decode an size metods
   int dr_code(char& var,int n_elements=1);
   int dr_code(double& var,int n_elements=1);
   int dr_code(float& var,int n_elements=1);
   int dr_code(int& var,int n_elements=1);
   int dr_code(long& var,int n_elements=1);
```

```
   int dr_code(short& var,int n_elements=1);
   int dr_code(unsigned char& var,int n_elements=1);
   int dr_code(unsigned int& var,int n_elements=1);
   int dr_code(unsigned long& var,int n_elements=1);
protected:
   int ldr_code(char& var,int n_elements);
   int ldr_code(double& var,int n_elements);
   int ldr_code(float& var,int n_elements);
   int ldr_code(int& var,int n_elements);
   int ldr_code(long& var,int n_elements);
   int ldr_code(short& var,int n_elements);
   int ldr_code(unsigned char& var,int n_elements);
   int ldr_code(unsigned int& var,int n_elements);
   int ldr_code(unsigned long& var,int n_elements);
virtual   int xdr_code(char& var,int n_elements);
virtual   int xdr_code(double& var,int n_elements);
virtual   int xdr_code(float& var,int n_elements);
virtual   int xdr_code(int& var,int n_elements);
virtual   int xdr_code(long& var,int n_elements);
virtual   int xdr_code(short& var,int n_elements);
virtual   int xdr_code(unsigned char& var,int n_elements);
virtual   int xdr_code(unsigned int& var,int n_elements);
virtual   int xdr_code(unsigned long& var,int n_elements);

};

=========================================================
=========================================================
```

```
=====================================================
              dll.h documentation
=====================================================
//
//(C) Copyright 1995, Diego Ernesto Malpica Chauvet.
//          All rights reserved.

#include "btypes.h"
#include "action.h"

template <class T>
class dll_node_C {
public:
  bt_C<dll_node_C> next,prev;
  T item;
};


template <class T>
class dll_C {
public:

  bt_C< dll_node_C<T> > f_ele,l_ele;
  bt_C< dll_node_C<T> > c_ele;
  int_B n_elements;
  int_B position;
  action_B action;
  void set_action (action_B& action);
  int  doit();//execute accion in the current element
  int  doit_all();//execute accion in all the elements
  int  is_out(void); //test if the pointer is out of the list
  void out(void); //set the pointer out of the list
  int get_location(void); //return the number of the current element
                          //return 0 if the pointer is out of the list
  int set_location(int location); //move the pointer to the location
  int size(void); // return the number of elements of the list
  int first(void); //move to the first element
  int last(void); //move to the last element
  int next(void); //move to the next element
  int prev(void); //move to the previous element
  int insert_r(T& item); // insert to the right of the curren element
                         // if is_out() insert as the first element
  int insert_l(T& item); // insert to the left of the current element
                         // if is_out() insert as the last element
  int del(void); // remove the current element
  void del_all(void); //remove all the elements
  T& item(void); //acces the item at the current element
            //you must be in a valid position !is_out()
  dll_C();
};

=====================================================
=====================================================
```

```
=====================================================
                dllk.h documentation
=====================================================
//
//(C) Copyright 1995, Diego Ernesto Malpica Chauvet.
//          All rights reserved.

#include "dll.h"
// T must be a clas tha implement a compare_k metod
// the compare_k metod shold return a integer at must have a
// (T& a,int key_op=1) signature (parameters).
// where a is the a referen with the object will be compare
// and key_op is a option for the compare

template <class T>
class dllk_C:public dll_C<T> {
public:
  int find     (T& x,int key=1);  //find, starting from the begining
  int find_o   (T& x,int key=1);  //find, in an order list from the begining
  int insert_o (T& x,int key=1);  //insert in order starting from the begining
  int insert_ou(T& x,int key=1);  //insert in order starting from the begining
                        //the key must be unique  if not no insertion
                        //is done
  int findb_o   (T& x,int key=1); //find, starting from the end
  int insertb_o (T& x,int key=1);//insert in order starting from the end
  int insertb_ou(T& x,int key=1);//insert in order starting from the end
                        //the key must be unique  if not no insertion
                        //is done
  //key is the is the option selector that will be used int compare_k metod
  //of the class t
  //insert metods return 1 in succes an 0 otherwise,
  //find metods return 1 if it have a match, 2 if it find a greater than and 0
  //otherwise (nothing finded).
  //findb metos return 1 if if have a macht, 2 if it find a less than and 0
  //otherwise (nothing finded).

};

=====================================================
=====================================================
```

```
=====================================================
                stack.h documentation
=====================================================
//
//(C) Copyright 1995, Diego Ernesto Malpica Chauvet.
//          All rights reserved.

#include "dll.h"

//Stack implemented using a double linked list

template <class T>
class stack_C:public dll_C<T> {
public:
  int  push(T& x);  //add an element on the top of the stack
  void pop();       //drop the elmente at the top of the stack
  T&   top();       //obtain the element at the top of the stack
};
=====================================================
=====================================================
```

```
======================================================
              storage.h documentation
======================================================
//
//(C) Copyright 1995, Diego Ernesto Malpica Chauvet.
//          All rights reserved.

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

class storage_C {
public:
  int open    (char* name,int mode_param=USE); //open the storage file
                                   //name -> filename
                                   //mode_param -> USE,CREATE
                                   //increment the open_count
  int close   (); // decrease open_count and close when open_count
               //reaches zero
  int stabilize();  //estabilize storage
  int remove   ();  //remove the file

  int get_rl(void* dato,u_long_B& pos,int level_param=-1,int trans_param=-1);
  //get and read lock
  int get_wl(void* dato,u_long_B& pos,int level_param=-1,int trans_param=-1);
  //get and write lock
  int get   (void* dato,u_long_B& pos,int level_param=-1,int trans_param=-1);
  //get
  int modify(void* dato,u_long_B& pos,int level_param=-1,int trans_param=-1);
  //modify and write lock
  int put   (void* dato,u_long_B& pos,int level_param=-1,int trans_param=-1);
  //insert a new object and write lock
  int del     (u_long_B& pos,int level_param=-1,int trans_param=-1);
  //delete an object del lock
  int first    (u_long_B& pos,int level=-1);//get the pos of the first element
  int last     (u_long_B& pos,int level=-1);//get the pos of the last element
  int next     (u_long_B& pos,int level=-1);//get the pos of the next element
  int prev     (u_long_B& pos,int level=-1);//get the pos of the prev element
  int first    (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  int last     (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  int next     (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  int prev     (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  int first_rl    (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  int last_rl     (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  int next_rl     (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  int prev_rl     (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  int first_wl    (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  int last_wl     (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  int next_wl     (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  int prev_wl     (void* dato,u_long_B& pos,int level=-1,int trans_param=-1);
  //trans_param is the number of the transaction
  //dato is a pointer to handler you are responsable that the pointer to a
  //a handler tha you pass is of a handler that could read the informantion
  //stored, if you are using named types you can read a sun from a parent
  //handler. (Polymorfic objects are suported).
  //all the medots that have de parameter dato do get of the object
  // the sufix rl stands for get and read lock
  // the sufix wl stands for get and  write lock
  //you can't get an object using  get if it is write locked
  //you can´t  get a read locked or a write loked whit a get and write lock
  //level is level in which the operation is do, levels are independents

  int set_root (u_long_B& pos,int level_param=-1,int trans_param=-1);
  //set root   and lock root
  int get_root (u_long_B& pos,int level_param=-1,int trans_param=-1);
  //get root   if root is not locked
  //level_param is the level in wich the operation is do

//                    Transactions
  int begin_trans();
  int end_trans(int trans_param=-1,char commit_f=1);
```

```
   // end the transaction if comit_f is 1 then the transsaction is commited
   // if commit_f is 0 the transaction is not comited
   int commit_trans(int trans_param);
   // if you commit a not ending transaction old the transaction is undone
   // else the transaction is consolidated
   int set_active_trans(int trans_param);//set the default transaction
   void set_level (int level=-1);//set the default level

//                     Initialization

   void init (int data_size=512,int default_level=1,int levels=5,int trans=10);
   //storage must be initialiazed with init metod
   //data_size is size of the data zone in each page
   //default level set the deafault level in the various metods -1 is used to
   //select the default level (active level(.
   //levels the number of levels available for user data
   //trans  the numbers of levels available for transactions this number
   //determine the number of simultaneus transactions


   storage_C();
   ~storage_C();
};

typedef bt_C<storage_C> storage_B; //type declaration of a handler

=======================================================
=======================================================
```

```
=====================================================
                key.h documentation
=====================================================
//
//(C) Copyright 1995, Diego Ernesto Malpica Chauvet.
//          All rights reserved.

#include "btypes.h"

class key_C {
public:
  string_B prefix;  //determines the key zone
  string_B key;     //key
  u_long_B recpos;  //offset in the file
  char_B   extra;   //field for extra information
  int compare_k (key_C& key,int key_opt=1); //compare with other key
     //key opt is used for select diferents compare options
     //first compare the prefix if they are not equal return >0 greater than
     //or <0 less than.
     //If the prefix are equals then compare return the result of compare the
     //keys
     //If opt is not 1 and the prefix and the keys are equals then return the
     //result of compare the recpos
  void clear(void); //clear all the files
  key_C(void);
};

typedef bt_C<key_C> key_B; //type declaration for a handler

=====================================================
=====================================================
```

```
=====================================================
                btree.h documentation
=====================================================
//
//(C) Copyright 1995, Diego Ernesto Malpica Chauvet.
//         All rights reserved.

#include "key.h"
#include "storage.h"
#include "dllk.h"
#include "stack.h"

class btree_C:public storage_C{
//btree_C support transcations, you should consider that the operations of
//btree_C consume an extra transaction level to make its operations atomics
//this is independent of your actual transaction
public:
  u_char_B level;  //this is the level used of the storage for the btree
                //you can change at any time to another level and you will
                //be using a diferent tree

  //the prefix of the key indicate a diferent idependent zone of the tree
  //this zone is independent for all the operations
  int first_key(key_B& key);//get the first key of the zone
  int last_key(key_B& key); //get the last key of the zone
  int prev_key(key_B& key); //get the prev key of the zone you must be in
                    //a valid position of the zone
  int next_key(key_B& key); //get the next key of the zone you must be in
                    //a valid position of the zone
  int find_key(key_B& key); //find a key in the zone return 1 if it had a match
                    //return 2 if it not have and exact match
  int find_exact_key(key_B& key); //find and exact match if it does'nt returns
                        //0, it uses the recpos pos in the key to
                        //find a especific object, this is important
                        //in the case that you are'nt using unic keys

  int add_key(key_B& key,int trans_param=-1); //add a key to the btree
  int del_key(key_B& key,int trans_param=-1); //delete a key of the btree

//                    Initialization
//you must initialize your btree before you use it whit

  void init(int data_size=512,int orden=10,int level=0);
//  this metod initialize the storage_C class with a datasize, the btree_C
// with an order orden, and set the level for the initial btree to 0
// all the others params of the btree_C and storage_C take the defaul values
// gived in the next two metods

  void init_btree(int orden=3, int level=0);
  void init_storage(int data_size=512,int default_level=1,int levels=5,
              int trans=10);
// you can use this two metods for a more especific initialization of the
// storage_C and btree_C class, you must use both.

};

typedef bt_C<btree_C> btree_B; //type declaration for a handler

=====================================================
=====================================================
```

===
except.h documentation
===

```
//symbolics constanst for the clasification of exceptions
//the data member info will have the code
//the metod msg return a char* that points to a strings with text information
//related with the exception.
//extra info data members are used to supply a frame work for codes and messages
//of the derived classes used for exceptions.
#define UNCLASIFIED_EXCEPT 0
#define MEMORY_EXCEPT 1
#define FILE_EXCEPT 2
#define TYPES_EXCEPT 3

class except_C {
public:
   int info;
   int extra_info;
   int extra_info2;
   virtual char* msg();
   virtual char* extra_msg();
   virtual char* extra_msg2();
   except_C(int info=0,int ext_info=0);
};

//Memory exceptions
#define UNKNOW_MEMORY_EXCEPT 0
#define NO_MEMORY_EXCEPT 1
class memory_except_C:public except_C{
public:
     char* msg();
     char* extra_msg();
     memory_except_C(int extra_info=0);
};

//File exceptions
#define UNKNOW_FILE_EXCEPT 0
#define OPEN_FILE_EXCEPT   1
#define CLOSE_FILE_EXCEPT  2
#define READ_FILE_EXCEPT   3
#define WRITE_FILE_EXCEPT  4
#define SEEK_FILE_EXCEPT   5

class file_except_C:public except_C{
public:
     char* msg();
     char* extra_msg();
     file_except_C(int extra_info=0);
};

//Types exceptions
#define UNKNOW_TYPES_EXCEPT 0
#define VOID_REFERENCE_TYPES_EXCEPT   1
#define TYPE_NOT_REGISTRED_TYPES_EXCEPT  2

class types_except_C:public except_C{
public:
     char* msg();
     char* extra_msg();
     types_except_C(int extra_info=0);
};
```

===
===