

# Using Bcfg in your Elfs

---

## Using Bcfg in your Elfs

by [Edgpaez](#) on 16 Mar 2009 01:42

Good day developers :grin:

Today you'll learn how to use and implement Bcfg in your elfs....the easiest way to start is using BcfgExample sources, you'll find those here:

[http://www.se-developers.net/configure- ... t-165.html](http://www.se-developers.net/configure-...t-165.html)

Now you'll find a lot of files, don't get scared, here's a little explanation :P

**conf\_loader.cpp:** I'm pretty sure the name says it all, here you'll find the functions to read the Bcfg file.

**conf\_loader.h:** Here are the declarations of the variables and functions implemented on .cpp file.

**config\_data.c:** Here is where you model how Bcfg is going to be, ( levels, strings, default values, etc ) and where you'll assign a variable to each configuration item.

**config\_data.h:** Here are declared the variables that will get their values from what's read on Bcfg. So here you define the variables you'll use in config\_data.c.

### **What you need to know:**

- Each configuration item:  
CFG\_UINT, CFG\_INT:

*[non-available image posted]*

CFG\_STR\_WIN1251, CFG\_UTF16\_STRING, CFG\_STR\_PASS:

*[non-available image posted]*

CFG\_CHECKBOX:

*[non-available image posted]*

CFG\_COORDINATES:

*[non-available image posted]*

CFG\_COLOR\_INT:

*[non-available image posted]*

CFG\_FONT:

*[non-available image posted]*

CFG\_KEYCODE:

*[non-available image posted]*

CFG\_RECT:

*[non-available image posted]*

**Thanks LiNkMaN for screens!**

- Each configuration item needs a different variable type:  
CFG\_UINT - unsigned int  
CFG\_INT - int  
CFG\_STR\_WIN1251 - char array  
CFG\_UTF16\_STRING - wchar\_t array  
CFG\_CBOX - int ( returns the position of the selected item )  
CFG\_STR\_PASS - wchar\_t array  
CFG\_COORDINATES - two ints ( x and y )  
CFG\_CHECKBOX - int ( 1 when check box is Selected and 0 if it isn't )  
CFG\_COLOR\_INT - unsigned int  
CFG\_TIME - TIME (Object)  
CFG\_DATE - DATE (Object)  
CFG\_FONT - int  
CFG\_KEYCODE - two ints ( key and mode )  
CFG\_RECT - RECT (Object)
- Don't forget the includes:  
In main:

Code: [Select all](#)

```
#include "..\\include\\cfg_items.h" //----> declaration of each Bcfg  
item.  
#include "conf_loader.h"  
#include "config_data.h"
```

In conf\_loader.cpp

Code: [Select all](#)

```
#include "conf_loader.h"
```

In config\_data.cpp

Code: [Select all](#)

```
#include "..\include\cfg_items.h"//----> declaration of each Bcfg
item.
#include "config_data.h"
```

- Remeber:

In main, ReconfigElf function will be the one to read the bcfg file when you changed it, so changes can be applied right after, so in main if it's not included you'll have to include:

Code: [Select all](#)

```
static int ReconfigElf(void *mess ,BOOK *book)
{
    RECONFIG_EVENT_DATA *reconf=(RECONFIG_EVENT_DATA *)mess;
    int result=0;
    if (wstrcmpr(reconf->path,succesed_config_path)==0 &&
wstrcmpr(reconf->name,succesed_config_name)==0)
    {
        InitConfig();
        Timer_ReSet(&timer,REFRESH_TIME,onTimer,0);
        result=1;
    }
    return(result);
}
```

## 1. Changing Bcfg file's name:

it's quite simple, in conf\_loader.cpp go to void InitConfig():

Code: [Select all](#)

```
void InitConfig(void)
{
    if (LoadConfigData(GetDir(DIR_ELFS_CONFIG|
MEM_EXTERNAL),L"BcfgExample.bcfg")<0)
    {
        LoadConfigData(GetDir(DIR_ELFS_CONFIG|
MEM_INTERNAL),L"BcfgExample.bcfg");
    }
}
```

Change the last parameter of both functions (L"BcfgExample.bcfg") to what ever you want ( no especial characters or spaces ) with bcfg extension, so that FileReg can use BcfgEdit to edit it:

Code: [Select all](#)

```
void InitConfig(void)
{
    if (LoadConfigData(GetDir(DIR_ELFS_CONFIG|
MEM_EXTERNAL), L"my_own_elf.bcfg")<0)
    {
        LoadConfigData(GetDir(DIR_ELFS_CONFIG|
MEM_INTERNAL), L"my_own_elf.bcfg");
    }
}
```

## 2. Declaring variables:

In config\_data.h you'll define all the variables you'll need for Bcfg, starting with "extern const" VARIABLE\_TYPE VARIABLE\_NAME; ( as I said before each item needs a different type so write the right ones ) ( VARIABLE\_TYPE = int, char[], wchar\_t[], char, unsigned int, .... )

Code: [Select all](#)

```
//Message one
extern const int ENABLED_1;
extern const wchar_t MESSAGE_1[256];
extern const unsigned int COLOR_1;
extern const int FONT_1;
extern const int MODE_1;
extern const int X_1;
extern const int Y_1;

//Message two
extern const int ENABLED_2;
extern const wchar_t MESSAGE_2[256];
extern const unsigned int COLOR_2;
extern const int FONT_2;
extern const int MODE_2;
extern const int X_2;
extern const int Y_2;;
```

## 3. Modeling Bcfg:

As known in config\_data.c you'll "make" your Bcfg, following this syntax:

```
"__root const CFG_HDR" ANY_NAME = { ITEM_NAME , "String for user" , Minimum(int) ,
Maximum(int) };
```

```
"__root const " VARIABLE_TYPE VARIABLE_NAME = ORIGINAL_VALUE;
```

if your Item is a CFG\_CBOX you'll need a third line like this:

```
"__root const CFG_CBOX_ITEM " ANY_NAME [Number of options ( int )] = { "String for first
value" , "String for second value" , "String for third value", ...};
```

- > This will create a single item in Bcfg, so for each item you need you'll have to re write that.
- > Minimum(int) = an int determining the minimum size of the item.
- > Maximum(int) = an int determining the maximum size of the item.

If you want to create a sub-Level you have to write this ( to open sub-level )

```
__root const CFG_HDR ANY_NAME = { CFG_LEVEL , "String to be shown for sub-level" ,
NUMBER_OF_LEVEL (int) ,0};
```

and to close:

```
__root const CFG_HDR ANY_NAME = { CFG_LEVEL , "" , 0 , 0};
```

->NUMBER\_OF\_LEVEL: First level should be 1, second two, third 3, .... otherwise level won't be shown.

Example ( following the variables in config\_data.h example):

Code: [Select all](#)

```
///----->First message<-----
__root const CFG_HDR cfghdr1={CFG_LEVEL,"Message one",1,0};//Creates
a sub-level for first message, in this level all the item will be
placed.
__root const CFG_HDR cfghdr2={CFG_CHECKBOX,"Enable",0,0}; //creates
a check box
__root const int ENABLED_1 = 1; //will make the checkbox selected
__root const CFG_HDR cfghdr3={CFG_UTF16_STRING,"Message",2,255};
//creates a space to write the message, message smaller than 255
characters and higher than 2
__root const wchar_t MESSAGE_1[256]=L"SE Developers";//Default value
message will be "SE Developers".
__root const CFG_HDR cfghdr4 ={CFG_COLOR_INT,"Color",0,0}; //creates
a color select item
__root const unsigned int COLOR_1 =0xFFFFFFFF; //defaul value will
be white
__root const CFG_HDR cfghdr33={CFG_FONT,"Font",0,0}; //Font
selection
__root const int FONT_1=0;
__root const CFG_HDR
cfghdr14={CFG_COORDINATES,"Position",0,0};//Coordinates selection
__root const int X_1; //first value will be X
__root const int Y_1; //second will be Y
__root const CFG_HDR cfghdr5 ={CFG_CBOX,"Show on",0,3}; //Radio
Buttons
__root const int MODE_1=0;//starts in first position
__root const CFG_CBOX_ITEM cfghdr6[3]= {"On music", "On lock",
"Always"}; //options
__root const CFG_HDR cfghdr7={CFG_LEVEL,"",0,0}; //closes level

///----->Second message<-----
__root const CFG_HDR cfghdr8={CFG_LEVEL,"Message two",2,0};//Creates
```

another level for second message, in this level all the item will be placed.

```
__root const CFG_HDR cfghdr9={CFG_CHECKBOX,"Enable",0,0}; //creates a check box
```

```
__root const int ENABLED_2 = 1; //will make the checkbox selected
```

```
__root const CFG_HDR
```

```
cfghdr10={CFG_UTF16_STRING,"Message",2,255}; //creates a space to write the message, message smaller than 255 characters and higher than 2
```

```
__root const wchar_t MESSAGE_2[256]=L"Bcfg Example";//Default value message will be "Bcfg Example".
```

```
__root const CFG_HDR cfghdr11={CFG_COLOR_INT,"Color",0,0};
```

```
//creates a color select item
```

```
__root const unsigned int COLOR_2 =0xFFFFFFFF; //defaul value will be white
```

```
__root const CFG_HDR cfghdr34={CFG_FONT,"Font",0,0}; //Font selection
```

```
__root const int FONT_2=0;
```

```
__root const CFG_HDR
```

```
cfghdr17={CFG_COORDINATES,"Position",0,0};//Coordinates selection
```

```
__root const int X_2; //first value will be X
```

```
__root const int Y_2;//second will be Y
```

```
__root const CFG_HDR cfghdr12={CFG_CBOX,"Show on",0,3}; //Radio Buttons
```

```
__root const int MODE_2=0;//starts in first position
```

```
__root const CFG_CBOX_ITEM cfghdr66[3]= {"On music", "On lock", "Always"}; //options
```

```
__root const CFG_HDR cfghdr13={CFG_LEVEL,"",0,0}; //closes level
```

I made a simple Mod of BcfgExample, I share explained sources so you can see everything working and how the code does it :grin: attached Project folder.

Hope you understood :grin: feel free to post any question, doubt or suggestion :P

**@Developers**

If you see any error, please correct me, I'm not good at this at all :grin:

Regards

**latest EcfgEdit elf: [BcfgEdit](#)**

**Attachments**

[BcfgExample.rar](#)  
(107.48 KiB)

## **RE: Using Bcfg in your Elfs**

by [Stonos](#) on 22 Mar 2009 21:18

Nice tutorial :)

It might be a good idea to include the unofficial BookManager Config event that will allow you to launch BcfgEdit for an elf using BookManager.

I don't remember by heart how you can do that, but I think that both MiniGPS (BCFG version obviously) and SimpleBiorhythm use this event.

---

## **RE: Using Bcfg in your Elfs**

by [symj](#) on 06 Aug 2009 17:37

hi

i have some question about bcfg configuration format.

it seems the bigger the configuration options the bigger the elf becomes.

now, if i don't know this if the bigger elf(size) = more heap consumption.

if this is true then would it be better to use ini file or some other method like softedit / eventedit.

and the minimum storage size is int = 4 bytes even if i don't need to use like

CFG\_CHECKBOX as int (4 bytes) to store a value of 0,1

now it can also be done as

CFG\_CHECKBOX as char (1 bytes) to store a value of 0,1

i hope i made my point clear.

and if i made a wrong statement please correct me as i'm new to bcfg configuration format.

regards