Config

Adam Dawes

COLLABORATORS			
	TITLE : Config		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Adam Dawes	July 1, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Config

1.1 Config v1.0

Config v1.0

by Adam Dawes

25th May 1996

1.2 Introduction

Well, I hate Windows as much as the next man, but occasionally I stumble across a good idea hidden away within the operating system.

There are a couple of functions buried in there for reading and writing configuration files, and they actually make things very easy. I decided I'd had enough of messing around with config files on my Amiga, so I've ported the functions to Amiga C.

The idea is that the configuration files take a definitive structure which the Config functions can understand. Each config file is split in to a

number of "Sections" (which are stored in the config file as a keyword inside square brackets). In each of these sections are a number of "Items", each of which contains an actual data item. The items are local to the section that contains them, so it's perfectly legal to use one item name in several sections, they'll all be treated separately.

The beauty behind the functions is you don't have to worry about creating files or scanning through them. Even when it comes to reading data, you don't have to care if the config file exists or not as you provide a default value to use if the file/section/item cannot be located. Everything is automated within the Config functions.

1.3 Using the Config Functions

There are 4 public functions within the Config source code, as follows:

WriteConfig()

WriteConfigNumber()

ReadConfig()

ReadConfigNumber()

Select one of the functions for further details.

1.4 WriteConfig()

```
Function:
    int WriteConfig(char *filename, char *section, char *item, char *data);

Parameters:
    filename = the name of the config file (eg, "S:MyConfig.cfg")
    section = the name of the section to add config data to
    item = the item within the section to contain the config data
    data = the actual data itself

Details:
    You do not need to worry about anything at all when calling this
    function. If the config file specified doesn't exist, it'll be
    created for you. If the section specified doesn't exist, that will
    be created. If the item doesn't exist, it'll be added to the
    appropriate section; if it does exist, the previous data will be
    replaced by the new data.

    If all goes well, the function will return CFG_WRITE_SUCCESS. If
    something goes wrong (run out of memory, or the specified file
    cannot be written to), CFG_WRITE_FAIL will be returned.
```

1.5 WriteConfigNumber()

```
Function:
    int WriteConfigNumber(char *filename, char *section, char *item, long data);

Parameters:
    filename = the name of the config file
    section = the name of the section to add config data to
    item = the item within the section to contain the config data
    data = a long integer to be written to the file

Details:
    This function is more useful when reading/writing numbers to the config
    file. It is actually only a small front-end to the WriteConfig()
    function. If you need to write numbers that aren't longs (floats, for
    instance), just make a copy of the WriteConfigNumber() function in to
    your own code, and change the datatypes around as necessary. You can
    then copy and alter ReadConfigNumber() appropriately as well to read
    the data back in.

    As before, this will return CFG_WRITE_SUCCESS is all is well, or
    CFG_WRITE_FAIL if something goes wrong.
```

1.6 ReadConfig()

```
Function:
    int ReadConfig(char *filename, char *section, char *item, char *buffer,
                   int bufsize, char *def);

Parameters:
    filename = the name of the config file
    section = the name of the section to read config data from
    item = the item within the section that contains the config data
    buffer = an empty character array ready to receive the config data
    bufsize = the maximum length of chars than can be written to the buffer
    def = a string that will be written to the buffer if the requested
          config item cannot be found

Details:
    Use ReadConfig() to get data back from your config file. It will look
    for the specified section/item in the specified file.

    If the file cannot be opened, the section cannot be located, or the
    item within the section is not found, the function will copy the
    default string to your buffer, and return CFG_READ_SUCCESS. Therefore,
    you don't need to care at all about whether the config file exists.
    Just tell the function what you want to receive if your requested data
    cannot be found.

    The function will return CFG_READ_SUCCESS if everything is ok, or
    CFG_READ_FAIL if the string to be returned is larger than the supplied
    buffer.
```

1.7 ReadConfigNumber()

```
Function:
    long ReadConfigNumber(char *filename, char *section, char *item,
         long def);

Parameters:
    filename = the name of the config file
    section = the name of the section to read config data from
    item = the item within the section that contains the config data
    def = a long value that will be returned if the requested config item
          cannot be found

Details:

    As with WriteConfigNumber(), this is just a front end to the
    ReadConfig() function.

    This will return as a long value, the number located in the data item
    specified. If the data item cannot be found, your default value will be
    returned instead.

    This function is assumed never to fail. The string buffer is quite
    large enough to hold a 32-bit value, so no errors should result using
    this function.
```

1.8 Requirements

```
Obviously, you will need a C compiler to make any use of these functions.
Config was written using DICE V2, but should work with any Amiga C
compiler.

The sourcecode is not currently portable to other platforms, as it uses the
list handling functions found in the exec.library.

The code should work with any version of the Amiga operating system, past
present and future.
```

1.9 Legal Stuff

```
Config and its associated files are not public domain. They may be
distributed freely as long as no unreasonable charge is imposed. They may
not be included within any commercial package without express written
permission from the author; the exceptions from this are the AmiNet CDs and
Fred Fish's collections. Config may only be distributed if all files
contained within the original archive are present.

I do not accept responsibility for any damage done to your system or data
lost, directly or indirectly, as a result from using this program or any of
its associated files. You use the program entirely at your own risk. Of
course if you *do* experience problems then I'll do what I can to sort them
out, and please let me know so that I can try to cure them in a future
```

release.

1.10 History

```
    v1.0 (25.5.96)

        Initial release.
```

1.11 Contacting the author

Please do write to me if you like Config or if you have any problems with it or suggestions for a new version. I can't promise to reply quickly if you write via snail-mail, but I will do my best to always reply to email messages. I can be contacted at:

InterNet
   Adam@beachyhd.demon.co.uk

   http://www.pavilion.co.uk/rda/adam

FidoNet
   Adam Dawes@2:441/93.5

SnailMail
   Adam Dawes
   47 Friar Road
   Brighton
   BN1 6NH
   England