Welcome to the ATRC C-CPP Documentation for version 2.2.0

- Home
- Installation guide
- API reference
- GitHub Project

Welcome to the ATRC C-CPP Documentation

This documentation provides everything you need to integrate, install, and utilize the ATRC library for your C and C++ projects. Whether you're looking for a detailed installation guide or an exhaustive API reference, you'll find all the necessary information here.

Get Started View API Reference Visit GitHub

Key Features

- Read and write resource files in C and C++
- Simple and intuitive API
- Lightweight and efficient
- Cross platform support
- Open source and free to use
- Helpful built-in functionalities, such as variable injection, preprocessor directives, and much more

About ATRC

ATRC is a cross platform C and C++ resource file parser that allows you to easily read and write resource files in your projects. It provides a simple and intuitive API that makes it easy to integrate with your existing codebase. ATRC is designed to be lightweight, efficient, and easy to use, making it the perfect choice for developers looking to add resource file support to their applications. Whether you're working on a game, a multimedia application, or any other type of software that requires resource file support, ATRC has you covered. With ATRC, you can quickly and easily load and save resource files, access individual resources, and manage resource data in a way that is both flexible and powerful. ATRC is open source and free to use, so you can start using it in your projects today without any restrictions.

Example resource file

```
#!ATRC
# DATA.ATRC
%operating_system%=Windows
%version%=10.0
%username%=placeholder

[userdata]
message_1=Hello, %username%!
message_2=Welcome to %operating_system% %version%!

# Injection: 0 = years, 1 = months, 2 = days, 3 = hours, 4 = minutes, 5 = seconds
message_3=Your current uptime is %*2*%.%*1*%.%*0*% %*3*%:%*4*%:%*5*%!
```

Example program for C++

```
// MAIN.CPP
#include <iostream>
#include <ATRC.h>
int main() {
    atrc::ATRC_FD fd = atrc::ATRC_FD("file.atrc");

    if(!fd.CheckStatus()) {
        std::cout << "File parsed unsuccesfully!" << std::endl;
        return 1;
    }

    std::cout << fd["var_name"] << std::endl;
    if(fd["block_name"]["key"] == "") {
        std::cout << "Key not found!" << std::endl;
    }

    std::string line = fd["block_name"]["key"];
    std::vector<std::string> args = {"Hello everyone"};
    std::cout << "Before: " << line << std::endl;
    std::string res = fd.InsertVar_S(line, args);
    std::cout << "After: " << res << std::endl;
    return 0;
}
```

Example program for C

```
// MAIN.C
#include "ATRC.h"
#include <stdio.h>
int main() {
    C_PATRC_FD filedata = Create_Empty_ATRC_FD();
    if (filedata == NULL) {
        printf("[FAIL] Create_Empty_ATRC_FD: Failed to create ATRC_FD\n");
        return 1;
    }

    if (!Read(filedata, "test.atrc", ATRC_READ_ONLY)) {
        printf("[FAIL] Read: Failed to read file 'test.atrc'\n");
        Destroy_ATRC_FD(filedata);
        return 1;
    }

    const char* varname = "test_variable";
    const char* value = ReadVariable(filedata, varname);
    if(value == NULL){
        printf("[FAIL] ReadVariable: Failed to read variable '%s'\n", varname);
    } else {
        printf("[PASS] ReadVariable: Value of '%s' is '%s'\n", varname, value);
    }

    const char* blockname = "test_block";
    if (AddBlock(filedata, blockname)) {
        printf("[PASS] AddBlock: Block '%s' added successfully\n", blockname);
    } else {
        printf("[FAIL] AddBlock: Failed to add block '%s'\n", blockname);
    }

    if (DoesExistBlock(filedata, blockname)) {
        printf("[PASS] DoesExistBlock: Block '%s' exists\n", blockname);
    } else {
        printf("[FAIL] DoesExistBlock: Block '%s' does not exist\n", blockname);
    }

    if (RemoveBlock(filedata, blockname)) {
        printf("[PASS] RemoveBlock: Block '%s' removed successfully\n", blockname);
    } else {
        printf("[FAIL] RemoveBlock: Failed to remove block '%s'\n", blockname);
    }

    Destroy_ATRC_FD(filedata);
    return 0;
}
```

Back to top

ATRC Header File Documentation, version 2.2.0

- Home
- Installation guide
- API reference
- GitHub Project

Overview

This document provides an overview of the `ATRC` header file, detailing the constants, macros, data structures, functions, and the ATRC standard library. Both C and C++ declarations are supported.

Table of Contents

Syntax

No names can contain `[`, `]`, `#`, `*`

- ## Header

  First row of the ATRC file must be `#!ATRC`, thus denoting the file as an ATRC file and the extension can be freely choosed.

- ## Insert/Inject marking

  Insert/Inject marking is a way to inject variables into strings.

  - ### %*%

    Injects are placed left to right, from 0 to n.

  - ### %*[index]%

    Injects are placed at the given index

  Examples

  ```
  #!ATRC
  [Block]
  # Injects are: {"World", ",", "!"}
  key=Hello%*% %*%%*%
  # After injection, output is "HelloWorld ,!"

  key2=Hello%*1% %*0%%*2%
  # After injection, output is "Hello, World!"
  ```

  Remarks

  See more: C: InsertVar_S, C++: InsertVar C++: InsertVar_S

- ## Reserved characters

  Reserved characters are used for special purposes in ATRC files.

  - ### #

    Denotes a comment. To use in a value, escape with a backslash. `\#`

  - ### %

    Denotes start of an inject or variable. To use in a value, escape with a backslash. `\%`

  - ### &

    Denotes whitespace. To use in a value, escape with a backslash. `\&`

    ```
    #!ATRC
    [block]
    key=hello
    # Outputs: "hello"

    key=&hello&
    # Outputs: " hello "
    ```

- ## Comments

  Comments are denoted by a `#` character.

- ## Variables

  Variables are defined as `%name%=value`. The name cannot contain `*`. All variables are constants. Reference variables in values with `%name%`.

  Example

```
#!ATRC
%name%=value
[block]
key=Reference to name: %name%
```

- **Blocks**

  Blocks are defined as `[name]`. They contain keys.

  Example

  ```
  #!ATRC
  [block]
  key=value
  ```

- **Keys**

  Keys are defined as `key=value`. They are contained within blocks.

  Example

  ```
  #!ATRC
  [block]
  key=value
  ```

- **Preprocessor directives**

  Preprocessor directives are currently under development.

Macros and Constants

- `ATRC_API`: Export/import macro for DLLs on Windows.
- `FILEHEADER`: File header constant (`"#!ATRC"`).

Data Structures

Global declarations

- **ReadMode**

  Enumeration for read modes.

  ```
  typedef enum ReadMode {
      ATRC_READ_ONLY,
      ATRC_CREATE_READ,
      ATRC_FORCE_READ,
  } ReadMode;
  ```

  - `ATRC_READ_ONLY`: Read from file.
  - `ATRC_CREATE_READ`: Create file if it doesn't exist and read it.
  - `ATRC_FORCE_READ`: Delete file if it exists, create it and read it.

    Remarks

    FILEHEADER is appended to the start of the file in `ATRC_CREATE_READ` and `ATRC_FORCE_READ`.

C Declarations

- **C_Variable**

  Structure for variables.

  ```
  typedef struct C_Variable {
      char *Name;
      char *Value;
      bool IsPublic;
  } C_Variable, *C_PVariable;
  ```

  - `Name`: Variable name.

    - `Value`: Variable value.
    - `IsPublic`: Visibility flag.

- **_C_Variable_Arr**

  Structure for variable arrays.

  ```
  typedef struct _C_Variable_Arr {
  C_Variable *Variables;
  uint64_t VariableCount;
  } C_Variable_Arr, *C_PVariable_Arr;
  ```

    - `Variables`: Array of variables.
    - `VariableCount`: Number of variables.

- **C_Key**

  Structure for keys.

  ```
  typedef struct C_Key {
      char *Name;
      char *Value;
  } C_Key, *C_PKey;
  ```

    - `Name`: Key name.
    - `Value`: Key value.

- **C_Block**

  Structure for blocks.

  ```
  typedef struct _C_Block {
      char *Name;
      C_Key *Keys;
      uint64_t KeyCount;
  } C_Block, *C_PBlock;
  ```

    - `Name`: Block name.
    - `Keys`: Array of keys.
    - `KeyCount`: Number of keys.

- **_C_Block_Arr**

  Structure for block arrays.

  ```
  typedef struct _C_Block_Arr {
      C_Block *Blocks;
      uint64_t BlockCount;
  } C_Block_Arr, *C_PBlock_Arr;
  ```

    - `Blocks`: Array of blocks.
    - `BlockCount`: Number of blocks.

- **_ATRCFiledata**

  Structure for ATRC file data.

  ```
  typedef struct _ATRCFiledata{
      C_PVariable_Arr Variables;
      C_PBlock_Arr Blocks;
      char *Filename;
      bool AutoSave;
      bool Writecheck;
  } C_ATRC_FD, *C_PATRC_FD;
  ```

    - `Variables`: Array of variables.
    - `Blocks`: Array of blocks.
    - `Filename`: File name.
    - `AutoSave`: Auto-save flag. Default: `false`.
    - `Writecheck`: If status is `true`, creates new keys, blocks or variables when modifying a value. Default: `false`

C++ Declarations

- ### Variable

  Structure for variables.

  ```
  typedef struct ATRC_API Variable {
      std::string Name;
      std::string Value;
      bool IsPublic = true;
  } Variable, * PVariable;
  ```

  - `Name`: Variable name.
  - `Value`: Variable value.
  - `IsPublic`: Visibility flag.

- ### Key

  Structure for keys.

  ```
  typedef struct ATRC_API Key {
      std::string Name;
      std::string Value;
  } Key, * PKey;
  ```

  - `Name`: Key name.
  - `Value`: Key value.

- ### _Block

  Structure for blocks.

  ```
  typedef struct ATRC_API _Block {
      std::string Name;
      std::vector Keys;
  } Block, * PBlock;
  ```

  - `Name`: Block name.
  - `Keys`: Vector of keys.

- ### ATRC_FD

  ATRC file data class

  ```
  class ATRC_API ATRC_FD {
      public:
          ATRC_FD();
          ATRC_FD(const char* path);
          ATRC_FD(C_PATRC_FD filedata);
          ~ATRC_FD();
          bool Read();
          std::string ReadVariable(const std::string& varname);
          std::string ReadKey(const std::string& block, const std::string& key);
          bool DoesExistBlock(const std::string& block);
          bool DoesExistVariable(const std::string& varname);
          bool DoesExistKey(const std::string& block, const std::string& key);
          bool IsPublic(const std::string& varname);
          void InsertVar(std::string& line, std::vector& args);
          std::string InsertVar_S(const std::string& line, std::vector& args);
          bool AddBlock(const std::string& blockname);
          bool RemoveBlock(const std::string& blockname);
          bool AddVariable(const std::string& varname, const std::string& value);
          bool RemoveVariable(const std::string& varname);
          bool ModifyVariable(const std::string& varname, const std::string& value);
          bool AddKey(const std::string& block, const std::string& key, const std::string& value);
          bool RemoveKey(const std::string& block, const std::string& key);
          bool ModifyKey(const std::string& block, const std::string& key, const std::string& value);
          C_PATRC_FD ToCStruct();

          bool CheckStatus();

          std::vector* GetVariables();
          std::vector* GetBlocks();
          std::string GetFilename();
          bool GetAutoSave() const;
          void SetAutoSave(bool autosave);
          bool GetWriteCheck() const;
          void SetWriteCheck(bool writecheck);
  ```

```
        PROXY_ATRC_FD operator[](const std::string& key);
        PROXY_ATRC_FD operator[](const std::string& key) const;

    private:
        void MAINCONSTRUCTOR();
        bool AutoSave;
        bool Writecheck;
        std::unique_ptr> Variables;
        std::unique_ptr> Blocks;
        std::string Filename;
    };
typedef ATRC_FD* PATRC_FD;
```

More on functions in the C++ Member Functions section.

- `AutoSave`: Auto-save flag. Default: `false`.
- `Writecheck`: If status is `true`, creates new keys, blocks or variables when modifying a value. Default: `false`
- `Variables`: Vector of variables.
- `Blocks`: Vector of blocks.
- `Filename`: File name.

- **PROXY_ATRC_FD**

  Proxy class for ATRC file data. Used for operator overloading.

```
class ATRC_API PROXY_ATRC_FD {
public:
    PROXY_ATRC_FD(ATRC_FD& fd, const std::string& key);
    PROXY_ATRC_FD operator[](const std::string& subKey);
    operator std::string() const;
    PROXY_ATRC_FD& operator=(const std::string& value);
    PROXY_ATRC_FD& operator>>(const std::string& value);
    PROXY_ATRC_FD& operator>>(const char* value);

    inline friend std::ostream& operator<<(std::ostream& os, const PROXY_ATRC_FD& obj) {
        uint64_t x = obj.key.find("]");
        if(x == std::string::npos) os << obj.fd->ReadVariable(obj.key);
        else {
            std::string block = obj.key.substr(0, x);
            std::string key_ = obj.key.substr(x + 1, obj.key.size() - x - 1);
            os << obj.fd->ReadKey(block, key_);
        }
        return os;
    }
private:
    PATRC_FD fd;
    std::string key;
};
```

More on functions in the C++ Member Functions section.

- `fd`: ATRC file data pointer.
- `key`: Key name.

Functions

C Functions

- **Read**

  Reads contents from a file

```
ATRC_API bool Read(C_PATRC_FD self, const char* path, ReadMode readMode = ATRC_READ_ONLY);
```

Arguments

- `filedata`: ATRC file data pointer.
- `filepath`: File path.
- `readMode`: Read mode.

Returns

- `true` if successful, `false` otherwise.

Remarks

Reads the contents of a file into an ATRC file data structure. If the function fails, the file data structure is cleared. and the function returns `false`, new data structure needs to be created. See ReadMode for more information on read modes.

- **ReadVariable**

  Reads a variable from a file

  ```
  char* ReadVariable(C_PATRC_FD filedata, const char* varname);
  ```

  Arguments

  - `filedata`: ATRC file data pointer.
  - `varname`: Variable name.

  Returns

  - Variable value.

  Remarks

  Reads the value of a variable from a file. Doesn't check if the variable is public or if it exists beforehand. If the variable does not exist, the function returns `NULL`. If unauthorized access is attempted, the function returns `NULL` and logs an error.

- **ReadKey**

  Reads a key from a file

  ```
  ATRC_API const char* ReadKey(C_PATRC_FD self, const char* block, const char* key);
  ```

  Arguments

  - `self`: ATRC file data pointer.
  - `block`: Block name.
  - `key`: Key name.

  Returns

  - Key value.

  Remarks

  Reads the value of a key from a file. Doesn't check if the key or block exists beforehand. If the key does not exist, the function returns `NULL`.

- **DoesExistBlock**

  Checks if a block exists in a file.

  ```
  ATRC_API bool DoesExistBlock(C_PATRC_FD self, const char* block);
  ```

  Arguments

  - `self`: ATRC file data pointer.
  - `block`: Block name.

  Returns

  - `true` if the block exists, `false` otherwise.

  Remarks

  Checks if a block exists in a file.

- **DoesExistVariable**

  Checks if a variable exists in a file.

  ```
  ATRC_API bool DoesExistVariable(C_PATRC_FD self, const char* varname);
  ```

  Arguments

  - `self`: ATRC file data pointer.
  - `varname`: Variable name.

  Returns

  - `true` if the variable exists, `false` otherwise.

  Remarks

  Checks if a variable exists in a file.

- **DoesExistKey**

  Checks if a key exists in a file.

  ```
  ATRC_API bool DoesExistKey(C_PATRC_FD self, const char* block, const char* key);
  ```

  Arguments

  - `self`: ATRC file data pointer.
  - `block`: Block name.
  - `key`: Key name.

  Returns

  - `true` if the key exists, `false` otherwise.
  - `false` if the block does not exist.

- **IsPublic**

  Checks if a variable is public.

  ```
  ATRC_API bool IsPublic(C_PATRC_FD self, const char* varname);
  ```

  Arguments

  - `self`: ATRC file data pointer.
  - `varname`: Variable name.

  Returns

  - `true` if the variable is public, `false` otherwise.

  Remarks

  Checks if a variable is public.

- **InsertVar_S**

  Inserts a value into a string containing inject markings. See Insert/Inject marking for syntax.

  ```
  ATRC_API char* InsertVar_S(const char* line, const char** args)
  ```

  Arguments

  - `line`: String containing inject markings.
  - `args`: Array of arguments, null terminated.

Returns

  - Heap allocated string with injected values.

Remarks

Inserts values into a string containing inject markings. Return value needs to be freed with. Insert array must be null terminated.

Example

```
const char* line = ReadVariable(fd, "test"); // "%*% %*%"
const char* args[] = { "test", "test", NULL };
char* res = InsertVar_S(line, args);
if (res == NULL) {
    printf("InsertVar_S: Failed to insert variables\n");
    return;
}
printf("InsertVar_S: '%s'\n", res); // Output: 'test test'
free(res);
```

- **AddBlock**

  Adds a block to a file.

  `ATRC_API bool AddBlock(C_PATRC_FD self, const char* blockname);`

  Arguments

  - `self`: ATRC file data pointer.
  - `blockname`: Block name.

  Returns

  - `true` if successful, `false` otherwise.

  Remarks

  Adds a block to a file. Checks if the block already exists before creating. If `AutoSave` is enabled, the block is added to the bottom of the file.

- **RemoveBlock**

  Removes a block from a file.

  `ATRC_API bool RemoveBlock(C_PATRC_FD self, const char* blockname);`

  Arguments

  - `self`: ATRC file data pointer.
  - `blockname`: Block name.

  Returns

  - `true` if successful, `false` otherwise.

  Remarks

  Removes a block from a file. Checks if the block exists before removing. If `AutoSave` is enabled, everything after the block is removed, until next block is found or EOF is encountered.

- **AddVariable**

  Adds a variable to a file.

  `ATRC_API bool AddVariable(C_PATRC_FD self, const char* varname, const char* value);`

  Arguments

  - `self`: ATRC file data pointer.
  - `varname`: Variable name.
  - `value`: Variable value.

Returns

  - `true` if successful, `false` otherwise.

Remarks

Adds a variable to a file. Checks if the variable already exists before creating. If `AutoSave` is enabled, the variable is added to the second line of the file.

- **RemoveVariable**

  Removes a variable from a file.

  ```
  ATRC_API bool RemoveVariable(C_PATRC_FD self, const char* varname);
  ```

  ##### Arguments

  - `self`: ATRC file data pointer.
  - `varname`: Variable name.

  ##### Returns

  - `true` if successful, `false` otherwise.

  ##### Remarks

  Removes a variable from a file. Checks if the variable exists before removing. If `AutoSave` is enabled, the variable is removed from the file.

- **ModifyVariable**

  Modifies a variable in a file.

  ```
  ATRC_API bool ModifyVariable(C_PATRC_FD self, const char* varname, const char* value);
  ```

  ##### Arguments

  - `self`: ATRC file data pointer.
  - `varname`: Variable name.
  - `value`: Variable value.

  ##### Returns

  - `true` if successful, `false` otherwise.

  ##### Remarks

  Modifies a variable in a file. Checks if the variable exists before modifying. If `AutoSave` is enabled, the variable is modified in the file.

- **AddKey**

  Adds a key to a block in a file.

  ```
  ATRC_API bool AddKey(C_PATRC_FD self, const char* block, const char* key, const char* value);
  ```

  ##### Arguments

  - `self`: ATRC file data pointer.
  - `block`: Block name.
  - `key`: Key name.
  - `value`: Key value.

  ##### Returns

- `true` if successful, `false` otherwise.

Remarks

Adds a key to a block in a file. Checks if the block exists before adding the key. Checks if the key already exists before creating. If `AutoSave` is enabled, the key is added to the block in the file.

- **RemoveKey**

  Removes a key from a block in a file.

  ```
  ATRC_API bool RemoveKey(C_PATRC_FD self, const char* block, const char* key);
  ```

  ##### Arguments

  - `self`: ATRC file data pointer.
  - `block`: Block name.
  - `key`: Key name.

  ##### Returns

  - `true` if successful, `false` otherwise.

  ##### Remarks

  Removes a key from a block in a file. Checks if the block and key exist before removing. If `AutoSave` is enabled, the key is removed from the block in the file.

- **ModifyKey**

  Modifies a key in a block in a file.

  ```
  ATRC_API bool ModifyKey(C_PATRC_FD self, const char* block, const char* key, const char* value);
  ```

  ##### Arguments

  - `self`: ATRC file data pointer.
  - `block`: Block name.
  - `key`: Key name.
  - `value`: Key value.

  ##### Returns

  - `true` if successful, `false` otherwise.

  ##### Remarks

  Modifies a key in a block in a file. Checks if the block and key exist before modifying. If `AutoSave` is enabled, the key is modified in the block in the file.

- **Create_ATRC_FD**

  Creates an ATRC file data structure from a file.

  ```
  ATRC_API C_PATRC_FD Create_ATRC_FD(char *filename, ReadMode readMode);
  ```

  ##### Arguments

  - `filename`: File name.
  - `readMode`: Read mode.

  ##### Returns

  - ATRC file data pointer.

  ##### Remarks

Creates an ATRC file data structure from a file. Free memory with Destroy_ATRC_FD. See ReadMode for more information on read modes.

- **Create_Empty_ATRC_FD**

  Creates an empty ATRC file data structure.

  ```
  ATRC_API C_PATRC_FD Create_Empty_ATRC_FD();
  ```

  Returns

  - ATRC file data pointer.

  Remarks

  Creates an empty ATRC file data structure. Free memory with Destroy_ATRC_FD.

- **Destroy_ATRC_FD_Blocks_And_Keys**

  Frees blocks and keys in an ATRC file data structure.

  ```
  ATRC_API void Destroy_ATRC_FD_Blocks_And_Keys(C_PATRC_FD filedata);
  ```

  Arguments

  - `filedata`: ATRC file data pointer.

  Remarks

  Frees blocks and keys in an ATRC file data structure.

- **Destroy_ATRC_FD_Variables**

  Frees variables in an ATRC file data structure.

  ```
  ATRC_API void Destroy_ATRC_FD_Variables(C_PATRC_FD filedata);
  ```

  Arguments

  - `filedata`: ATRC file data pointer.

  Remarks

  Frees variables in an ATRC file data structure.

- **Destroy_ATRC_FD**

  Frees an ATRC file data structure.

  ```
  ATRC_API void Destroy_ATRC_FD(C_PATRC_FD filedata);
  ```

  Arguments

  - `filedata`: ATRC file data pointer.

  Remarks

  Frees an ATRC file data structure.

C++ Member Functions

ATRC_FD

- **ATRC_FD**

  Default constructor.

```
ATRC_FD();
```

Remarks

Initializes an ATRC file data structure.

- **ATRC_FD**

  Constructor with file path.

  ```
  ATRC_FD(const char* path, ReadMode mode = ATRC_READ_ONLY);
  ```

  Arguments

  - `path`: File path.
  - `mode`: Read mode.

  Remarks

  Initializes an ATRC file data structure and reads the file. See ReadMode for more information on read modes.

- **ATRC_FD**

  Constructor with C structure.

  ```
  ATRC_FD(C_PATRC_FD filedata);
  ```

  Arguments

  - `filedata`: C structure pointer.

  Remarks

  Initializes an ATRC file data structure from a C structure. C structure needs to be freed manually.

- **~ATRC_FD**

  Destructor.

  ```
  ~ATRC_FD();
  ```

  Remarks

  Frees an ATRC file data structure.

- **Read**

  Reads a file.

  ```
  bool Read(ReadMode mode);
  ```

  Arguments

  - `mode`: Read mode.

  Returns

  - `true` if successful, `false` otherwise.

  Remarks

  Reads a file into an ATRC file data structure. See ReadMode for more information on read modes.

- **ReadVariable**

Reads a variable from a file.

```
std::string ReadVariable(const std::string& varname);
```

Arguments

- `varname`: Variable name.

Returns

- Variable value.

Remarks

Reads the value of a variable from a file. Doesn't check if the variable is public or if it exists beforehand. If the variable does not exist, the function returns `""`. If unauthorized access is attempted, the function returns `""` and logs an error.

- **ReadKey**

  Reads a key from a file.

  ```
  std::string ReadKey(const std::string& block, const std::string& key);
  ```

  Arguments

  - `block`: Block name.
  - `key`: Key name.

  Returns

  - Key value.

  Remarks

  Reads the value of a key from a file. Doesn't check if the key or block exists beforehand. If the key does not exist, the function returns `""`.

- **DoesExistBlock**

  Checks if a block exists in a file.

  ```
  bool DoesExistBlock(const std::string& block);
  ```

  Arguments

  - `block`: Block name.

  Returns

  - `true` if the block exists, `false` otherwise.

  Remarks

  Checks if a block exists in a file.

- **DoesExistVariable**

  Checks if a variable exists in a file.

  ```
  bool DoesExistVariable(const std::string& varname);
  ```

  Arguments

  - `varname`: Variable name.

  Returns

    - `true` if the variable exists, `false` otherwise.

  Remarks

  Checks if a variable exists in a file.

- **DoesExistKey**

  Checks if a key exists in a file.

  ```
  bool DoesExistKey(const std::string& block, const std::string& key);
  ```

  Arguments

    - `block`: Block name.
    - `key`: Key name.

  Returns

    - `true` if the key exists, `false` otherwise.
    - `false` if the block does not exist.

- **IsPublic**

  Checks if a variable is public.

  ```
  bool IsPublic(const std::string& varname);
  ```

  Arguments

    - `varname`: Variable name.

  Returns

    - `true` if the variable is public, `false` otherwise.

  Remarks

  Checks if a variable is public.

- **InsertVar**

  Inserts a value into a string containing inject markings. See Insert/Inject marking for syntax.

  ```
  void InsertVar(std::string& line, std::vector& args);
  ```

  Arguments

    - `line`: String containing inject markings.
    - `args`: Vector of arguments.

  Remarks

  Inserts values into a string containing inject markings. Result is stored in `line`.

- **InsertVar_S**

  Inserts a value into a string containing inject markings. See Insert/Inject marking for syntax.

  ```
  std::string InsertVar_S(const std::string& line, std::vector& args);
  ```

  Arguments

    - `line`: String containing inject markings.
    - `args`: Vector of arguments.

Returns

  - String with injected values.

Remarks

Inserts values into a string containing inject markings.

- **AddBlock**

  Adds a block to a file.

  ```
  bool AddBlock(const std::string& blockname);
  ```

  Arguments

    - `blockname`: Block name.

  Returns

    - `true` if successful, `false` otherwise.

  Remarks

  Adds a block to a file. Checks if the block already exists before creating. If `AutoSave` is enabled, the block is added to the bottom of the file.

- **RemoveBlock**

  Removes a block from a file.

  ```
  bool RemoveBlock(const std::string& blockname);
  ```

  Arguments

    - `blockname`: Block name.

  Returns

    - `true` if successful, `false` otherwise.

  Remarks

  Removes a block from a file. Checks if the block exists before removing. If `AutoSave` is enabled, everything after the block is removed, until next block is found or EOF is encountered.

- **AddVariable**

  Adds a variable to a file.

  ```
  bool AddVariable(const std::string& varname, const std::string& value);
  ```

  Arguments

    - `varname`: Variable name.
    - `value`: Variable value.

  Returns

    - `true` if successful, `false` otherwise.

  Remarks

  Adds a variable to a file. Checks if the variable already exists before creating. If `AutoSave` is enabled, the variable is added to the second line of the file.

- **RemoveVariable**

  Removes a variable from a file.

  ```
  bool RemoveVariable(const std::string& varname);
  ```

  Arguments

  - `varname`: Variable name.

  Returns

  - `true` if successful, `false` otherwise.

  Remarks

  Removes a variable from a file. Checks if the variable exists before removing. If `AutoSave` is enabled, the variable is removed from the file.

- **ModifyVariable**

  Modifies a variable in a file.

  ```
  bool ModifyVariable(const std::string& varname, const std::string& value);
  ```

  Arguments

  - `varname`: Variable name.
  - `value`: Variable value.

  Returns

  - `true` if successful, `false` otherwise.

  Remarks

  Modifies a variable in a file. Checks if the variable exists before modifying. If `AutoSave` is enabled, the variable is modified in the file.

- **AddKey**

  Adds a key to a block in a file.

  ```
  bool AddKey(const std::string& block, const std::string& key, const std::string& value);
  ```

  Arguments

  - `block`: Block name.
  - `key`: Key name.
  - `value`: Key value.

  Returns

  - `true` if successful, `false` otherwise.

  Remarks

  Adds a key to a block in a file. Checks if the block exists before adding the key. Checks if the key already exists before creating. If `AutoSave` is enabled, the key is added to the block in the file.

- **RemoveKey**

  Removes a key from a block in a file.

  ```
  bool RemoveKey(const std::string& block, const std::string& key);
  ```

  Arguments

- `block`: Block name.
- `key`: Key name.

Returns

- `true` if successful, `false` otherwise.

Remarks

Removes a key from a block in a file. Checks if the block and key exist before removing. If `AutoSave` is enabled, the key is removed from the block in the file.

- **ModifyKey**

  Modifies a key in a block in a file.

  ```
  bool ModifyKey(const std::string& block, const std::string& key, const std::string& value);
  ```

  Arguments

  - `block`: Block name.
  - `key`: Key name.
  - `value`: Key value.

  Returns

  - `true` if successful, `false` otherwise.

  Remarks

  Modifies a key in a block in a file. Checks if the block and key exist before modifying. If `AutoSave` is enabled, the key is modified in the block in the file.

- **ToCStruct**

  Converts an ATRC file data structure to a C structure.

  ```
  C_PATRC_FD ToCStruct();
  ```

  Returns

  - C structure pointer.

  Remarks

  Converts an ATRC file data structure to a C structure. C structure needs to be freed manually.

- **CheckStatus**

  Checks the status of the ATRC file data class instance.

  ```
  bool CheckStatus();
  ```

  Returns

  - `true` if the file is parsed succesfully, `false` otherwise.

  Remarks

  Checks the status of the ATRC_FD class instance.

- **GetVariables**

  Gets a list of variables in a file.

```
std::vector GetVariables();
```

Returns

- Vector of variable names.

Remarks

Gets a list of variables in a file.

- **GetBlocks**

  Gets a list of blocks in a file.

  ```
  std::vector GetBlocks();
  ```

  Returns

  - Vector of block names.

  Remarks

  Gets a list of blocks in a file.

- **GetFilename**

  Gets the file name.

  ```
  std::string GetFilename();
  ```

  Returns

  - File name.

  Remarks

  Gets the file name.

- **GetAutoSave**

  Gets the AutoSave setting.

  ```
  bool GetAutoSave();
  ```

  Returns

  - `true` if AutoSave is enabled, `false` otherwise.

  Remarks

  Gets the AutoSave setting.

- **SetAutoSave**

  Sets the AutoSave setting.

  ```
  void SetAutoSave(bool autosave);
  ```

  Arguments

  - `autosave`: AutoSave setting.

  Remarks

  Sets the AutoSave setting.

- **GetWriteCheck**

  Gets the WriteCheck setting.

  ```
  bool GetWriteCheck() const;
  ```

  Returns

  - Status of `Writecheck`.

  Remarks

  Gets the WriteCheck setting. See ATRC_FD for more information.

- **SetWriteCheck**

  Sets the WriteCheck setting.

  ```
  void SetWriteCheck(bool writecheck);
  ```

  Remarks

  Sets the WriteCheck setting. See ATRC_FD for more information.

- **operator[]**

  Gets a variable or key value.

  ```
  PROXY_ATRC_FD operator[](const std::string& key);
  ```

  Arguments

  - `key`: Key name.

  Returns

  - Variable or key value.

  Remarks

  Gets a variable or key value.

- **operator[]**

  Gets a block.

  ```
  PROXY_ATRC_FD operator[](const std::string& key) const;
  ```

  Arguments

  - `key`: Block name.

  Returns

  - Block.

  Remarks

  Gets a block.

PROXY_ATRC_FD

- **PROXY_ATRC_FD**

  Default constructor.

```
PROXY_ATRC_FD(ATRC_FD& fd, const std::string& key);
```

Arguments

- `fd`: ATRC file data structure.
- `key`: Key name.

Remarks

Initializes a proxy ATRC file data structure.

- **operator[]**

  Gets a variable or key value.

  ```
  PROXY_ATRC_FD operator[](const std::string& subKey);
  ```

  Arguments

  - `subKey`: Key name.

  Returns

  - Variable or key value.

  Remarks

  Gets a variable or key value.

- **operator std::string**

  Converts the value to a string.

  ```
  operator std::string() const;
  ```

  Returns

  - Value as a string.

  Remarks

  Converts the value to a string.

- **operator const char***

  Converts the value to a C string.

  ```
  operator const char*() const;
  ```

  Returns

  - Value as a C string.

  Remarks

  Converts the value to a C string.

- **operator=**

  Assigns a value to a variable or key.

  ```
  PROXY_ATRC_FD& operator=(const std::string& value);
  ```

  Arguments

  - `value`: Value to assign.

Returns

  - Reference to the proxy ATRC file data structure.

Remarks

Assigns a value to a variable or key.

- **operator>>**

  Assigns a value to a variable or key.

  ```
  PROXY_ATRC_FD& operator>>(const std::string& value);
  ```

  Arguments

  - `value`: Value to assign.

  Returns

  - Reference to the proxy ATRC file data structure.

  Remarks

  Assigns a value to a variable or key.

- **operator>>**

  Assigns a value to a variable or key.

  ```
  PROXY_ATRC_FD& operator>>(const char* value);
  ```

  Arguments

  - `value`: Value to assign.

  Returns

  - Reference to the proxy ATRC file data structure.

  Remarks

  Assigns a value to a variable or key.

- **operator<<**

  Outputs a variable or key value.

  ```
  inline friend std::ostream& operator<<(std::ostream& os, const PROXY_ATRC_FD& obj) {
      uint64_t x = obj.key.find("]");
      if(x == std::string::npos) os << obj.fd->ReadVariable(obj.key);
      else {
          std::string block = obj.key.substr(0, x);
          std::string key_ = obj.key.substr(x + 1, obj.key.size() - x - 1);
          os << obj.fd->ReadKey(block, key_);
      }
      return os;
  }
  ```

  Arguments

  - `os`: Output stream.
  - `obj`: Proxy ATRC file data structure.

  Returns

    - Output stream.

  Remarks

  Outputs a variable or key value.

Operator Overloading

- **Reading a value**

  `operator[]`

  Remarks

  Gets a variable or key value.

  Example

  ```
  ATRC_FD fd("file.atrc");
  std::string var = fd["variable"];
  std::string key = fd["block"]["key"];
  ```

- **Assigning a value**

  `operator=`

  Remarks

  Assigns a value to a variable or key.

  Example

  ```
  ATRC_FD fd("file.atrc");
  fd["variable"] = "value";
  fd["block"]["key"] = "value";
  ```

- **Appending a value**

  `operator>>`

  Remarks

  Appends a value to a variable or key.

  Example

  ```
  ATRC_FD fd("file.atrc");
  fd["variable"] >> "value";
  fd["block"]["key"] >> "value";
  ```

- **Outputting a value**

  `operator<<`

  Remarks

  Outputs a variable or key value.

  Example

  ```
  ATRC_FD fd("file.atrc");
  std::cout << fd["variable"] << std::endl;
  std::cout << fd["block"]["key"] << std::endl;
  ```

ATRC Standard Library

Overview

The ATRC standard library provides utility functions for data conversion and manipulation.

Data structures, enumerations, and global variables

- **ATRC_ERR**

  Error enumeration.

  ```
  enum ATRC_ERR {
      _ATRC_SUCCESSFULL_ACTION = 0,
      _ATRC_UNSUCCESSFULL_ACTION = 1
  };
  ```

  #### Values

  - `_ATRC_SUCCESSFULL_ACTION`: Successful action.
  - `_ATRC_UNSUCCESSFULL_ACTION`: Unsuccessful action.

  #### Remarks

  Error enumeration.

- **atrc_stdlib_errval**

  Global error value.

  ```
  extern uint64_t atrc_stdlib_errval;
  ```

  #### Remarks

  Global error value, used to store the last error code for stdlib functions. Every stdlib function sets this value, _ATRC_SUCCESSFULL_ACTION if succesfull, _ATRC_UNSUCCESSFULL_ACTION otherwise.

- **_C_String_Arr**

  Array structure for C strings.

  ```
  typedef struct _C_String_Arr {
      char **list;
      uint64_t count;
  } C_String_Arr, *C_PString_Arr;
  ```

  #### Members

  - `list`: Array of C strings.
  - `count`: Number of strings in the array.

  #### Remarks

  Array structure for C strings.

Functions

- **atrc_to_vector**

  Converts a string to a vector of strings.

  ```
  ATRC_API std::vector<std::string> atrc_to_vector(char separator, const std::string &value);
  ```

  #### Arguments

  - `separator`: Separator character.
  - `value`: String to convert.

  #### Returns

- Vector of strings.

Remarks

Converts a string to a vector of strings.

- **atrc_to_list**

  Converts a string to a list of strings.

  ```
  ATRC_API C_PString_Arr atrc_to_list(char separator, const char* value);
  ```

  Arguments

  - `separator`: Separator character.
  - `value`: String to convert.

  Returns

  - Array of C strings.

  Remarks

  Converts a string to a list of strings. Free the array with stdlib_functions_atrc_free_list

- **atrc_free_list**

  Frees a list of strings.

  ```
  ATRC_API void atrc_free_list(C_PString_Arr list);
  ```

  Arguments

  - `list`: Array of C strings.

  Remarks

  Frees a list of strings.

- **atrc_to_bool**

  Converts a string to a boolean.

  ```
  ATRC_API bool atrc_to_bool(const char* value);
  ```

  Arguments

  - `value`: String to convert.

  Returns

  - Boolean value.

  Remarks

  Converts a string to a boolean. Accepts `"true|TRUE"`, `"false|FALSE"`, `"1"`, `"0"`.

- **atrc_to_uint64_t**

  Converts a string to an unsigned 64-bit integer.

  ```
  ATRC_API uint64_t atrc_to_uint64_t(const char* value);
  ```

  Arguments

  - `value`: String to convert.

Returns

- Unsigned 64-bit integer.

Remarks

Converts a string to an unsigned 64-bit integer.

- **atrc_to_int64_t**

  Converts a string to a signed 64-bit integer.

  ```
  ATRC_API int64_t atrc_to_int64_t(const char* value);
  ```

  Arguments

  - `value`: String to convert.

  Returns

  - Signed 64-bit integer.

  Remarks

  Converts a string to a signed 64-bit integer.

- **atrc_to_double**

  Converts a string to a double.

  ```
  ATRC_API double atrc_to_double(const char* value);
  ```

  Arguments

  - `value`: String to convert.

  Returns

  - Double.

  Remarks

  Converts a string to a double.

[Back to top]

ATRC Header File Documentation, version 2.2.0

- Home
- Installation guide
- API reference
- GitHub Project

Installation Guide

Welcome to the ATRC installation guide! Follow the steps below to install and integrate ATRC into your project.

Step 1: Download ATRC

You can download the latest release of ATRC from the official GitHub repository:

- All Releases
- Latest Release

Once downloaded, extract the contents of the archive to a directory of your choice.

Step 2: Set Up Your Project with CMake

Use the following example to configure your project with CMake to include the ATRC library:

```
cmake_minimum_required(VERSION 3.15)
project(MyProject)

list(APPEND CMAKE_MODULE_PATH "${CMAKE_SOURCE_DIR}/ATRC_2.2.0/cmake")

# Include the ATRCConfig.cmake script
include(ATRCConfig.cmake)

add_executable(${PROJECT_NAME}
    src/main.cpp
)

# Link the ATRC library
target_link_libraries(${PROJECT_NAME} PRIVATE ${ATRC})

# Include ATRC headers
target_include_directories(${PROJECT_NAME} PRIVATE "${CMAKE_SOURCE_DIR}/ATRC_2.2.0/include")
```

Replace `ATRC_2.2.0` with the version number of the ATRC release you downloaded if it's different.

Step 3: Build and Run

After configuring your project with CMake, build the project using your preferred build system:

1. Run `cmake` to generate the build files:

   ```
   cmake -S . -B build
   ```

2. Build the project:

   ```
   cmake --build build
   ```

3. Run the executable from the `build` directory:

   ```
   ./build/MyProject
   ```

Additional Notes

- Ensure that your compiler and environment support C++17 or later.
- Refer to the API Reference for details on how to use ATRC's features in your application.
- If you encounter any issues, visit the GitHub Issues page to report bugs or ask for help.

Back to top