

ZInitFile 클래스 사용 가이드

개요

ZInitFile 은 UTF-8 인코딩을 지원하는 INI 파일 파서 클래스입니다. Windows API의 GetPrivateProfileString 대신 표준 C++ std::ifstream / std::ofstream 을 사용하여 UTF-8 한글을 포함한 모든 유니코드 문자를 올바르게 처리합니다.

특징

- ✔ UTF-8 완벽 지원 - 한글, 중국어, 일본어 등 모든 유니코드 문자
- ✔ 캐시 기반 - 파일을 메모리에 캐시하여 빠른 읽기/쓰기
- ✔ 표준 INI 형식 - [Section] 및 Key:Value (또는 Key=Value) 구조
- ✔ BOM 처리 - UTF-8 BOM 자동 감지 및 제거
- ✔ 유연한 API - 섹션명 또는 인덱스 기반 접근

파일 형식

기본 구조

```
# 주석 ( ';' 또는 '#'로 시작)
[SectionName]
Key1: Value1
Key2: Value2

[AnotherSection]
Key3: Value3
```

ZGUI에서 사용하는 형식 (.ift)

```
[InitFile]
FontRes: ./Data/FontRes.ift
TextureRes: ./Data/TextureRes.ift

[0]
FontName: ./Data/Fonts/Arial_16.spritefont
Size: 16
Bold: 0
Italic: 0

[1]
FontName: ./Data/Fonts/NanumGothic_12.spritefont
Size: 12
Bold: 0
Italic: 0
```

클래스 API

생성자 및 소멸자

```
ZInitFile();
~ZInitFile();
```

사용 예:

```
ZInitFile iniFile;
```

파일 로드

```
BOOL LoadIFT(const char* pFilePath)
```

```
BOOL LoadIFT(const std::string& filePath)
```

파일을 로드하고 내부 캐시에 저장합니다.

매개변수:

- pFilePath / filePath: INI 파일 경로

반환값:

- TRUE: 성공
- FALSE: 실패 (파일 없음, 읽기 오류)

사용 예:

```
ZInitFile iniFile;

// C 스타일 문자열
if (iniFile.LoadIFT("./Data/Settings.ift"))
{
    std::cout << "파일 로드 성공!" << std::endl;
}

// std::string
std::string path = "./Data/Config.ift";
if (iniFile.LoadIFT(path))
{
    std::cout << "파일 로드 성공!" << std::endl;
}
```

값 읽기

```
std::string GetValue(const std::string& section, const std::string& key, const std::string& defaultValue = "")
```

```
std::string GetValue(const char* pSection, const char* pKey, const char* pDefault = "")
```

섹션과 키로 값을 읽습니다.

매개변수:

- section / pSection: 섹션 이름
- key / pKey: 키 이름
- defaultValue / pDefault: 키가 없을 때 반환할 기본값

반환값:

- 키에 해당하는 값 (문자열)
- 키가 없으면 기본값

사용 예:

```
// 기본값 없음
std::string fontPath = iniFile.GetValue("InitFile", "FontRes");

// 기본값 지정
std::string language = iniFile.GetValue("Settings", "Language", "Korean");
```

```
// 한글 값 읽기 (UTF-8)
std::string title = iniFile.GetValue("UI", "WindowTitle", "기본 제목");
```

```
std::string GetValue(int iIndex, const std::string& key, const std::string& defaultValue = "")
```

```
std::string GetValue(int iIndex, const char* pKey, const char* pDefault = "")
```

인덱스 기반 섹션에서 값을 읽습니다. 섹션 이름이 "0", "1", "2" 등인 경우 사용합니다.

매개변수:

- `iIndex`: 섹션 인덱스 (정수)
- `key` / `pKey`: 키 이름
- `defaultValue` / `pDefault`: 기본값

사용 예:

```
// FontRes.ift
// [0]
// FontName: Arial_16.spritefont
// Size: 16

std::string fontName = iniFile.GetValue(0, "FontName");
std::string fontSize = iniFile.GetValue(0, "Size", "12");

// 여러 폰트 로드
for (int i = 0; i < 10; i++)
{
    if (iniFile.HasSection(i))
    {
        std::string name = iniFile.GetValue(i, "FontName");
        std::string size = iniFile.GetValue(i, "Size");
        // 폰트 추가...
    }
}
```

값 쓰기

```
BOOL SetValue(const std::string& section, const std::string& key, const std::string& value)
```

```
BOOL SetValue(const char* pSection, const char* pKey, const char* pValue)
```

```
BOOL SetValue(int iIndex, const char* pKey, const char* pValue)
```

섹션과 키에 값을 씁니다. 캐시와 파일 모두 업데이트됩니다.

매개변수:

- `section` / `pSection` / `iIndex`: 섹션 이름 또는 인덱스
- `key` / `pKey`: 키 이름
- `value` / `pValue`: 설정할 값

반환값:

- `TRUE`: 성공
- `FALSE`: 실패

사용 예:

```
// 새 값 설정
iniFile.SetValue("Settings", "Language", "English");
```

```
// 한글 값 쓰기 (UTF-8)
iniFile.SetValue("UI", "WindowTitle", "새 창 제목");

// 인덱스 기반
iniFile.SetValue(0, "FontName", "NewFont.spritefont");

// 숫자 값 (문자열로 변환 필요)
iniFile.SetValue("Player", "Score", std::to_string(1000));
```

주의사항:

- SetValue 호출 시 즉시 파일에 기록됩니다
- 섹션이 없으면 자동으로 생성됩니다
- 키가 없으면 추가되고, 있으면 업데이트됩니다

섹션 관리

```
BOOL GetTitleList(std::vector<std::string>& outList)
```

```
BOOL GetSectionList(std::vector<std::string>& outList)
```

모든 섹션 이름을 가져옵니다. (두 메서드는 동일)

매개변수:

- outList : 섹션 이름을 저장할 벡터 (출력 매개변수)

반환값:

- TRUE : 성공
- FALSE : 실패

사용 예:

```
std::vector<std::string> sections;
if (iniFile.GetTitleList(sections))
{
    std::cout << "섹션 목록:" << std::endl;
    for (const auto& section : sections)
    {
        std::cout << "  [" << section << "]" << std::endl;
    }
}

// 숫자 섹션만 처리
for (const auto& sectionName : sections)
{
    try
    {
        int index = std::stoi(sectionName);
        // 숫자 섹션 처리
        std::string value = iniFile.GetValue(index, "SomeKey");
    }
    catch (const std::exception&)
    {
        // 숫자가 아닌 섹션 무시
        continue;
    }
}
```

```
BOOL GetKeyList(const char* pSection, std::vector<std::string>& outList)
```

특정 섹션의 모든 키를 가져옵니다.

매개변수:

- pSection : 섹션 이름
- outList : 키 이름을 저장할 벡터

사용 예:

```
std::vector<std::string> keys;
if (iniFile.GetKeyList("Settings", keys))
{
    std::cout << "[Settings] 섹션의 키:" << std::endl;
    for (const auto& key : keys)
    {
        std::string value = iniFile.GetValue("Settings", key);
        std::cout << "  " << key << " = " << value << std::endl;
    }
}
```

BOOL HasSection(const char* pSection)

BOOL HasSection(int iIndex)

섹션이 존재하는지 확인합니다.

사용 예:

```
if (iniFile.HasSection("Settings"))
{
    std::cout << "[Settings] 섹션 존재" << std::endl;
}

// 인덱스 기반
if (iniFile.HasSection(0))
{
    std::cout << "[0] 섹션 존재" << std::endl;
}
```

유틸리티

const std::string& GetFilePath() const

로드된 파일의 경로를 반환합니다.

```
std::cout << "현재 파일: " << iniFile.GetFilePath() << std::endl;
```

BOOL IsLoaded() const

파일이 로드되었는지 확인합니다.

```
if (iniFile.IsLoaded())
{
    std::cout << "파일이 로드됨" << std::endl;
}
```

void Clear()

캐시를 비우고 파일을 언로드합니다.

```
iniFile.Clear();
```

완전한 사용 예제

예제 1: 설정 파일 읽기

Settings.ift:

```
[Graphics]
Width: 1920
Height: 1080
Fullscreen: 1

[Audio]
MasterVolume: 100
MusicVolume: 80
SFXVolume: 90

[Player]
Name: 플레이어1
Level: 5
```

C++ 코드:

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

int main()
{
    ZInitFile settings;

    // 파일 로드
    if (!settings.LoadIFT("./Settings.ift"))
    {
        std::cerr << "설정 파일 로드 실패!" << std::endl;
        return 1;
    }

    // 그래픽 설정 읽기
    int width = std::stoi(settings.GetValue("Graphics", "Width", "800"));
    int height = std::stoi(settings.GetValue("Graphics", "Height", "600"));
    bool fullscreen = std::stoi(settings.GetValue("Graphics", "Fullscreen", "0")) != 0;

    std::cout << "해상도: " << width << "x" << height << std::endl;
    std::cout << "전체화면: " << (fullscreen ? "예" : "아니오") << std::endl;

    // 오디오 설정 읽기
    int masterVol = std::stoi(settings.GetValue("Audio", "MasterVolume", "100"));
    std::cout << "마스터 볼륨: " << masterVol << "%" << std::endl;

    // 플레이어 정보 읽기 (UTF-8 한글)
    std::string playerName = settings.GetValue("Player", "Name", "Unknown");
    int playerLevel = std::stoi(settings.GetValue("Player", "Level", "1"));

    std::cout << "플레이어: " << playerName << " (레벨 " << playerLevel << ")" << std::endl;

    return 0;
}
```

예제 2: 리소스 파일 로드 (ZGUI 방식)

FontRes.ift:

```
[0]
FontName: ./Data/Fonts/Arial_16.spritefont
Size: 16
Bold: 0
Italic: 0

[1]
FontName: ./Data/Fonts/NanumGothic_12.spritefont
Size: 12
Bold: 0
Italic: 0

[2]
FontName: ./Data/Fonts/Impact_24_Bold.spritefont
Size: 24
Bold: 1
Italic: 0
```

C++ 코드:

```
void LoadFonts(ZGUIResource* pResource, const std::string& fontResPath)
{
    ZInitFile fontRes;

    if (!fontRes.LoadIFT(fontResPath))
    {
        std::cerr << "FontRes.ift 로드 실패!" << std::endl;
        return;
    }

    // 모든 섹션 가져오기
    std::vector<std::string> sections;
    fontRes.GetTitleList(sections);

    // 숫자 섹션만 처리
    for (const auto& sectionName : sections)
    {
        try
        {
            int index = std::stoi(sectionName);

            // 폰트 정보 읽기
            std::string fontName = fontRes.GetValue(index, "FontName");
            int size = std::stoi(fontRes.GetValue(index, "Size"));
            int bold = std::stoi(fontRes.GetValue(index, "Bold"));
            int italic = std::stoi(fontRes.GetValue(index, "Italic"));

            // 폰트 추가
            pResource->AddFont(fontName, size, bold, italic);

            std::cout << "폰트 로드: " << fontName << " (크기: " << size << ")" << std::endl;
        }
        catch (const std::exception&)
        {
            // 숫자가 아닌 섹션 무시
            continue;
        }
    }
}
```

예제 3: 설정 파일 쓰기

```
void SaveSettings(int width, int height, bool fullscreen, int volume)
{
    ZInitFile settings;

    // 기존 파일 로드 (있으면)
    settings.LoadIFT("./Settings.ift");
```

```
// 새 값 설정
settings.SetValue("Graphics", "Width", std::to_string(width));
settings.SetValue("Graphics", "Height", std::to_string(height));
settings.SetValue("Graphics", "Fullscreen", fullscreen ? "1" : "0");
settings.SetValue("Audio", "MasterVolume", std::to_string(volume));

std::cout << "설정 저장 완료!" << std::endl;
}

// 사용
SaveSettings(1920, 1080, true, 85);
```

예제 4: 모든 섹션과 키 나열

```
void PrintAllData(const std::string& filePath)
{
    ZInitFile iniFile;

    if (!iniFile.LoadIFT(filePath))
    {
        std::cerr << "파일 로드 실패: " << filePath << std::endl;
        return;
    }

    // 모든 섹션 가져오기
    std::vector<std::string> sections;
    iniFile.GetTitleList(sections);

    std::cout << "=== " << filePath << " ===" << std::endl;

    for (const auto& section : sections)
    {
        std::cout << "\n[" << section << "]" << std::endl;

        // 섹션의 모든 키 가져오기
        std::vector<std::string> keys;
        if (iniFile.GetKeyList(section.c_str(), keys))
        {
            for (const auto& key : keys)
            {
                std::string value = iniFile.GetValue(section, key);
                std::cout << "  " << key << ": " << value << std::endl;
            }
        }
    }
}
```

.ift 파일 생성

방법 1: 프로그래밍 방식으로 생성

기본 파일 생성

```
void CreateSettingsFile()
{
    ZInitFile settings;

    // 기본 경로 설정 (파일은 SetValue 호출 시 자동 생성됨)
    // 먼저 빈 파일로 LoadIFT를 호출하거나 직접 SetValue 사용

    // Graphics 섹션
    settings.SetValue("Graphics", "Width", "1920");
    settings.SetValue("Graphics", "Height", "1080");
    settings.SetValue("Graphics", "Fullscreen", "1");
    settings.SetValue("Graphics", "VSync", "1");
```

```

// Audio 섹션
settings.SetValue("Audio", "MasterVolume", "100");
settings.SetValue("Audio", "MusicVolume", "80");
settings.SetValue("Audio", "SFXVolume", "90");

// Player 섹션 (UTF-8 한글)
settings.SetValue("Player", "Name", "플레이어1");
settings.SetValue("Player", "Level", "1");
settings.SetValue("Player", "Experience", "0");

std::cout << "Settings.ift 파일 생성 완료!" << std::endl;
}

```

ZGUI 리소스 파일 생성

```

void CreateFontResFile()
{
    ZInitFile fontRes;

    // 폰트 0
    fontRes.SetValue(0, "FontName", "./Data/Fonts/Arial_16.spritefont");
    fontRes.SetValue(0, "Size", "16");
    fontRes.SetValue(0, "Bold", "0");
    fontRes.SetValue(0, "Italic", "0");

    // 폰트 1
    fontRes.SetValue(1, "FontName", "./Data/Fonts/NanumGothic_12.spritefont");
    fontRes.SetValue(1, "Size", "12");
    fontRes.SetValue(1, "Bold", "0");
    fontRes.SetValue(1, "Italic", "0");

    // 폰트 2
    fontRes.SetValue(2, "FontName", "./Data/Fonts/Impact_24_Bold.spritefont");
    fontRes.SetValue(2, "Size", "24");
    fontRes.SetValue(2, "Bold", "1");
    fontRes.SetValue(2, "Italic", "0");

    std::cout << "FontRes.ift 파일 생성 완료!" << std::endl;
}

void CreateTextureResFile()
{
    ZInitFile textureRes;

    // 텍스처 0
    textureRes.SetValue(0, "FileName", "./Data/Textures/button_normal.png");
    textureRes.SetValue(0, "TransparentA", "255");
    textureRes.SetValue(0, "TransparentR", "255");
    textureRes.SetValue(0, "TransparentG", "0");
    textureRes.SetValue(0, "TransparentB", "255");

    // 텍스처 1
    textureRes.SetValue(1, "FileName", "./Data/Textures/dialog_bg.png");
    textureRes.SetValue(1, "TransparentA", "255");
    textureRes.SetValue(1, "TransparentR", "0");
    textureRes.SetValue(1, "TransparentG", "0");
    textureRes.SetValue(1, "TransparentB", "0");

    std::cout << "TextureRes.ift 파일 생성 완료!" << std::endl;
}

void CreateDialogResFile()
{
    ZInitFile dialogRes;

    // 메인 다이얼로그
    dialogRes.SetValue(0, "DialogName", "MainDialog");
    dialogRes.SetValue(0, "X", "100");
    dialogRes.SetValue(0, "Y", "100");
    dialogRes.SetValue(0, "Width", "400");
    dialogRes.SetValue(0, "Height", "300");
}

```

```

dialogRes.SetValue(0, "Keyboard", "1");
dialogRes.SetValue(0, "Mouse", "1");
dialogRes.SetValue(0, "Dragable", "1");
dialogRes.SetValue(0, "Visible", "1");

std::cout << "DialogRes.ift 파일 생성 완료!" << std::endl;
}

void CreateControlResFile()
{
    ZInitFile controlRes;

    // 버튼 0 (시작 버튼)
    controlRes.SetValue(0, "DialogName", "MainDialog");
    controlRes.SetValue(0, "Type", "2"); // ZGUI_CONTROL_BUTTON
    controlRes.SetValue(0, "Text", "시작하기");
    controlRes.SetValue(0, "FontID", "0");
    controlRes.SetValue(0, "TextureID", "0");
    controlRes.SetValue(0, "AlignHorizontal", "1"); // Center
    controlRes.SetValue(0, "AlignVertical", "1"); // Center
    controlRes.SetValue(0, "dstLeft", "150");
    controlRes.SetValue(0, "dstTop", "200");
    controlRes.SetValue(0, "dstRight", "250");
    controlRes.SetValue(0, "dstBottom", "240");
    controlRes.SetValue(0, "srcLeft", "0");
    controlRes.SetValue(0, "srcTop", "0");
    controlRes.SetValue(0, "srcRight", "100");
    controlRes.SetValue(0, "srcBottom", "40");
    controlRes.SetValue(0, "Vertical", "1");
    controlRes.SetValue(0, "Default", "0");
    controlRes.SetValue(0, "Enable", "1");
    controlRes.SetValue(0, "Visible", "1");

    // 레이블 1 (제목)
    controlRes.SetValue(1, "DialogName", "MainDialog");
    controlRes.SetValue(1, "Type", "0"); // ZGUI_CONTROL_LABEL
    controlRes.SetValue(1, "Text", "게임 제목");
    controlRes.SetValue(1, "FontID", "1");
    controlRes.SetValue(1, "AlignHorizontal", "1");
    controlRes.SetValue(1, "AlignVertical", "1");
    controlRes.SetValue(1, "dstLeft", "100");
    controlRes.SetValue(1, "dstTop", "50");
    controlRes.SetValue(1, "dstRight", "300");
    controlRes.SetValue(1, "dstBottom", "80");
    controlRes.SetValue(1, "Default", "0");
    controlRes.SetValue(1, "Enable", "1");
    controlRes.SetValue(1, "Visible", "1");

    std::cout << "ControlRes.ift 파일 생성 완료!" << std::endl;
}

```

모든 리소스 파일 한 번에 생성

```

void InitializeAllResourceFiles()
{
    // DefaultRes.ift 생성
    ZInitFile defaultRes;
    defaultRes.SetValue("InitFile", "FontRes", "./Data/FontRes.ift");
    defaultRes.SetValue("InitFile", "TextureRes", "./Data/TextureRes.ift");
    defaultRes.SetValue("InitFile", "DialogRes", "./Data/DialogRes.ift");
    defaultRes.SetValue("InitFile", "ControlRes", "./Data/ControlRes.ift");
    std::cout << "DefaultRes.ift 생성 완료!" << std::endl;

    // 나머지 리소스 파일들 생성
    CreateFontResFile();
    CreateTextureResFile();
    CreateDialogResFile();
    CreateControlResFile();

    std::cout << "\n모든 리소스 파일 생성 완료!" << std::endl;
}

```

```
// 사용
int main()
{
    // Data 폴더 생성 (필요한 경우)
    CreateDirectory(L"./Data", NULL);
    CreateDirectory(L"./Data/Fonts", NULL);
    CreateDirectory(L"./Data/Textures", NULL);

    // 리소스 파일 생성
    InitializeAllResourceFiles();

    return 0;
}
```

방법 2: 텍스트 에디터로 직접 작성

Visual Studio Code / Notepad++

1. 새 파일 생성
 - 파일 이름: `Settings.ift`
 - 인코딩: **UTF-8** 또는 **UTF-8 with BOM**
2. 내용 작성

```
[Graphics]
Width: 1920
Height: 1080
Fullscreen: 1

[Audio]
MasterVolume: 100
MusicVolume: 80

[Player]
Name: 플레이어1
Level: 5
```

3. 저장
 - Visual Studio Code: 우측 하단에서 인코딩 확인 (UTF-8)
 - Notepad++: `인코딩` 메뉴 → `UTF-8` 선택

Windows 메모장

주의: Windows 메모장은 UTF-8 BOM을 기본으로 추가하지 않으므로 한글이 깨질 수 있습니다.

해결 방법:

1. PowerShell 사용:

```
$content = @"
[Graphics]
Width: 1920
Height: 1080
"@

$Utf8NoBomEncoding = New-Object System.Text.UTF8Encoding $False
[System.IO.File]::WriteAllLines("Settings.ift", $content, $Utf8NoBomEncoding)
```

2. 또는 Visual Studio Code / Notepad++ 사용 권장

방법 3: 템플릿으로부터 복사

템플릿 파일 생성

Template.ift:

```
# ZGUI Resource Template
# Encoding: UTF-8

[0]
FontName: ./Data/Fonts/YourFont.spritefont
Size: 16
Bold: 0
Italic: 0

[1]
FontName: ./Data/Fonts/YourFont2.spritefont
Size: 12
Bold: 0
Italic: 0
```

C++로 템플릿 복사 및 수정

```
void CreateFromTemplate(const std::string& templatePath, const std::string& outputPath)
{
    ZInitFile iniFile;

    // 템플릿 로드
    if (!iniFile.LoadIFT(templatePath))
    {
        std::cerr << "템플릿 로드 실패: " << templatePath << std::endl;
        return;
    }

    // 값 수정
    iniFile.SetValue(0, "FontName", "./Data/Fonts/CustomFont.spritefont");
    iniFile.SetValue(0, "Size", "20");

    // 새 파일로 저장되지 않으므로, 파일을 복사하거나
    // 또는 모든 값을 SetValue로 다시 설정해야 함

    std::cout << "템플릿으로부터 파일 생성 완료!" << std::endl;
}
```

설정 파일 자동 생성 (First Run)

```
class GameSettings
{
private:
    ZInitFile m_Settings;
    std::string m_FilePath;

public:
    GameSettings(const std::string& filePath) : m_FilePath(filePath)
    {
        // 파일이 없으면 기본 설정 생성
        if (!m_Settings.LoadIFT(filePath))
        {
            CreateDefaultSettings();
            m_Settings.LoadIFT(filePath);
        }
    }

    void CreateDefaultSettings()
    {
        std::cout << "기본 설정 파일 생성 중..." << std::endl;

        ZInitFile defaultSettings;

        // Graphics
        defaultSettings.SetValue("Graphics", "Width", "1920");
```

```
defaultSettings.SetValue("Graphics", "Height", "1080");
defaultSettings.SetValue("Graphics", "Fullscreen", "0");
defaultSettings.SetValue("Graphics", "VSync", "1");
defaultSettings.SetValue("Graphics", "AntiAliasing", "1");

// Audio
defaultSettings.SetValue("Audio", "MasterVolume", "100");
defaultSettings.SetValue("Audio", "MusicVolume", "80");
defaultSettings.SetValue("Audio", "SFXVolume", "90");
defaultSettings.SetValue("Audio", "VoiceVolume", "100");

// Controls
defaultSettings.SetValue("Controls", "MouseSensitivity", "50");
defaultSettings.SetValue("Controls", "InvertY", "0");

// Game
defaultSettings.SetValue("Game", "Difficulty", "Normal");
defaultSettings.SetValue("Game", "Language", "Korean");

std::cout << "기본 설정 파일 생성 완료: " << m_FilePath << std::endl;
}

int GetInt(const std::string& section, const std::string& key, int defaultValue)
{
    return std::stoi(m_Settings.GetValue(section, key, std::to_string(defaultValue)));
}

std::string GetString(const std::string& section, const std::string& key, const std::string& defaultValue)
{
    return m_Settings.GetValue(section, key, defaultValue);
}

void SetInt(const std::string& section, const std::string& key, int value)
{
    m_Settings.SetValue(section, key, std::to_string(value));
}

void SetString(const std::string& section, const std::string& key, const std::string& value)
{
    m_Settings.SetValue(section, key, value);
}
};

// 사용
int main()
{
    GameSettings settings("./Settings.ift");

    // 설정 읽기
    int width = settings.GetInt("Graphics", "Width", 800);
    int height = settings.GetInt("Graphics", "Height", 600);
    std::string language = settings.GetString("Game", "Language", "English");

    std::cout << "해상도: " << width << "x" << height << std::endl;
    std::cout << "언어: " << language << std::endl;

    // 설정 변경
    settings.SetInt("Graphics", "Width", 2560);
    settings.SetString("Game", "Language", "Japanese");

    return 0;
}
```

내부 구조

캐시 메커니즘

```
// 내부 캐시 구조
std::map<std::string, std::map<std::string, std::string>> m_Cache;
```

```
// 섹션 이름 → (키 → 값)
```

동작 방식:

- 1. LoadIFT() 호출 시 파일을 읽어 m_Cache 에 저장
- 2. GetValue() 호출 시 캐시에서 직접 읽기 (파일 I/O 없음)
- 3. SetValue() 호출 시 캐시 업데이트 + 파일 전체 재작성

장점:

- 빠른 읽기 성능 (O(log n))
- 메모리에서 작업하므로 파일 I/O 최소화

단점:

- SetValue() 시 전체 파일 재작성 (큰 파일은 느릴 수 있음)

UTF-8 처리

Windows 콘솔 출력 설정

ZInitFile은 UTF-8을 올바르게 처리하지만, Windows 콘솔에서 한글을 표시하려면:

```
// ZInitFile::LoadCache()에서 자동 설정됨
#ifdef _WIN32
SetConsoleOutputCP(CP_UTF8);
#endif
```

또는 프로그램 시작 시:

```
int main()
{
    // Windows 콘솔을 UTF-8 모드로 설정
    SetConsoleOutputCP(CP_UTF8);

    ZInitFile iniFile;
    iniFile.LoadIFT("./한글파일.ift");

    std::cout << iniFile.GetValue("섹션", "키", "기본값") << std::endl;
}
```

주의사항

1. 파일 형식

```
# ✔ 올바른 형식
[Section]
Key: Value

# ✔ 등호 사용 가능 (현재 콜론으로 설정됨)
[Section]
Key = Value

# ✖ 잘못된 형식
Key: Value      # 섹션 없음 (무시됨)

[Section
Key: Value      # 닫는 괄호 없음
```

```
[  
Key: Value      # 빈 섹션명
```

2. 타입 변환

```
// ❌ 잘못된 방법  
int value = iniFile.GetValue("Section", "Key"); // 컴파일 오류  
  
// ✅ 올바른 방법  
int value = std::stoi(iniFile.GetValue("Section", "Key", "0"));  
float fValue = std::stof(iniFile.GetValue("Section", "Float", "1.0"));  
bool bValue = (std::stoi(iniFile.GetValue("Section", "Bool", "0")) != 0);
```

3. 기본값 제공

```
// ✅ 항상 기본값 제공 권장  
std::string value = iniFile.GetValue("Section", "Key", "DefaultValue");  
  
// ❌ 기본값 없으면 빈 문자열 반환  
std::string value = iniFile.GetValue("Section", "Key");  
int num = std::stoi(value); // 빈 문자열이면 예외 발생!
```

4. 예외 처리

```
try  
{  
    int value = std::stoi(iniFile.GetValue("Section", "Key"));  
}  
catch (const std::exception& e)  
{  
    std::cerr << "변환 실패: " << e.what() << std::endl;  
    // 기본값 사용  
}
```

성능 팁

1. 한 번만 로드

```
// ✅ 좋은 방법  
ZInitFile settings;  
settings.LoadIFT("./Settings.ift");  
  
for (int i = 0; i < 1000; i++)  
{  
    std::string value = settings.GetValue("Section", "Key"); // 캐시에서 읽기  
}  
  
// ❌ 나쁜 방법  
for (int i = 0; i < 1000; i++)  
{  
    ZInitFile settings;  
    settings.LoadIFT("./Settings.ift"); // 매번 파일 읽기!  
    std::string value = settings.GetValue("Section", "Key");  
}
```

2. SetValue 최소화

```
// ❌ 나쁜 방법 (파일을 100번 재작성)  
for (int i = 0; i < 100; i++)  
{  
    iniFile.SetValue("Scores", std::to_string(i), std::to_string(i * 10));  
}
```

```
// ✅ 더 나은 방법 (필요할 때만 저장)
// 메모리에서 작업하고 마지막에 한 번만 저장
```

문제 해결

문제: 한글이 깨짐

원인: 파일이 UTF-8이 아님
해결: 파일을 UTF-8 BOM 또는 UTF-8 without BOM으로 저장

문제: 값을 읽지 못함

- 원인:
- 섹션/키 이름 오타
 - 파일 형식 오류 (콜론 대신 등호 사용)
 - 공백 처리 문제

해결:

```
// 디버그 출력
std::vector<std::string> sections;
iniFile.GetTitleList(sections);
for (const auto& s : sections)
{
    std::cout << "Section: [" << s << "]" << std::endl;
}
```

문제: SetValue가 작동하지 않음

원인: 파일 쓰기 권한 없음
해결: 파일 권한 확인, 관리자 권한으로 실행

관련 클래스

- ZGUIManager**: ZInitFile을 사용하여 리소스 파일 로드
- ZGUIResource**: 폰트, 텍스처, 다이얼로그 인덱스 관리

작성일: 2025-11-01
버전: 1.0
작성자: Cascade AI