**sctPluginTracker.h**

#pragma once

#include <vector>

#include <string>

#include "SmartPointer.h"

#include "sctPlugin.h"

#include <filesystem> // Добавлено для работы с файловой системой

class sctPluginTracker {

public:

// Получение единственного экземпляра sctPluginTracker

static sctPluginTracker\* GetInstance();

// Загрузка конфигурации плагинов из XML файла

void LoadPluginConfigurationXML(const char\* xmlFilePath);

// Установка пути к папке с плагинами

void SetPluginsPath(const std::string& path);

// Загрузка конфигурации плагинов из стандартного файла .plugins

void LoadDefaultPluginsConfiguration();

// Загрузка плагинов из выбранного файла

void LoadPluginsFromSelectedFile(const std::string& fileName);

// Получение количества плагинов

unsigned int GetNumberOfPlugins();

// Регистрация доступного плагина по имени файла

unsigned int RegisterAvailablePlugin(const std::string& filename);

// Регистрация плагина

void RegisterPlugin(sctPlugin\* plugin);

// Получение плагина по индексу

sctPlugin\* GetPlugin(unsigned int index);

// Получение имени плагина по индексу

std::string GetPluginName(unsigned int index);

// Получение имени файла плагина по индексу

std::string GetPluginFileName(unsigned int index);

// Проверка, загружен ли плагин по индексу

bool GetPluginLoaded(unsigned int index);

// Проверка, установлен ли автозагрузчик плагина по индексу

bool GetPluginAutoLoad(unsigned int index);

// Получение списка файлов в директории

std::vector<std::string> GetFilesInDirectory(const std::string& directory);

// Выбор файла из списка

std::string SelectFile(const std::vector<std::string>& files);

private:

sctPluginTracker() = default;

~sctPluginTracker() = default;

struct PluginInfo {

std::string m\_FileName;

std::string m\_PluginName;

sctPlugin\* m\_Plugin = nullptr;

bool m\_Autoload = false;

};

std::vector<PluginInfo> m\_PluginsList;

static SmartPointer<sctPluginTracker> Instance;

std::string m\_PluginsPath;

// Локализация плагина по имени плагина

std::vector<PluginInfo>::iterator locateUsingPluginName(const std::string& pluginName);

// Локализация плагина по имени файла

std::vector<PluginInfo>::iterator locateUsingFileName(const std::string& fileName);

};

// Поиск плагина по имени файла

std::string sctLocatePluginSerial(const std::string& pluginName);

// Получение имени плагина из имени файла

std::string sctGetPluginNameFromFileName(const std::string& filename);

**Реализация класса sctPluginTracker в sctPluginTracker.cpp**

**Замена vtkPVPluginTrackerDebugMacro на std::cout для отладки**

#include "sctPluginTracker.h"

#include "PluginLoader.h"

#include "SystemTools.h"

#include "SmartPointer.h"

#include "sctPlugin.h"

#include <fstream>

#include <sstream>

#include <iostream>

#include <filesystem>

namespace fs = std::filesystem;

// Реализация функции для поиска плагина по имени

std::string sctLocatePluginSerial(const std::string& pluginName) {

std::string app\_dir = sctBaseAppliccation::Instance().GetApplicationPath();

std::vector<std::string> paths\_to\_search;

paths\_to\_search.push\_back(app\_dir);

std::string name = pluginName;

std::string filename = name;

#if defined(\_WIN32) && !defined(\_\_CYGWIN\_\_)

filename = name + ".dll";

#else

filename = "lib" + name + ".so";

#endif

for (size\_t cc = 0; cc < paths\_to\_search.size(); cc++) {

std::string path = paths\_to\_search[cc];

if (sctSystemTools::FileExists((path + "/" + filename).c\_str(), true)) {

return (path + "/" + filename);

}

}

return std::string();

}

// Получение имени плагина из имени файла

std::string sctGetPluginNameFromFileName(const std::string& filename) {

std::string defaultname = sctSystemTools::GetFileNameWithoutExtension(filename);

if (defaultname.size() > 3 && filename.substr(0, 3) == "lib") {

defaultname.erase(0, 3);

}

return defaultname;

}

// Локализация плагина по имени плагина

std::vector<sctPluginTracker::PluginInfo>::iterator sctPluginTracker::locateUsingPluginName(const std::string& pluginName) {

for (auto iter = m\_PluginsList.begin(); iter != m\_PluginsList.end(); ++iter) {

if (iter->m\_PluginName == pluginName) {

return iter;

}

}

return m\_PluginsList.end();

}

// Локализация плагина по имени файла

std::vector<sctPluginTracker::PluginInfo>::iterator sctPluginTracker::locateUsingFileName(const std::string& fileName) {

for (auto iter = m\_PluginsList.begin(); iter != m\_PluginsList.end(); ++iter) {

if (iter->m\_FileName == fileName) {

return iter;

}

}

return m\_PluginsList.end();

}

// Создание нового экземпляра sctPluginTracker

sctPluginTracker\* sctPluginTracker::New() {

sctPluginTracker\* ret = new sctPluginTracker;

ret->InitializeObjectBase();

return ret;

}

// Инициализация статического указателя на экземпляр sctPluginTracker

SmartPointer<sctPluginTracker> sctPluginTracker::Instance = nullptr;

// Получение единственного экземпляра sctPluginTracker

sctPluginTracker\* sctPluginTracker::GetInstance() {

if (!Instance.GetPointer()) {

sctPluginTracker\* mgr = new sctPluginTracker();

Instance = mgr;

mgr->FastDelete();

}

return Instance.GetPointer();

}

// Установка пути к папке с плагинами

void sctPluginTracker::SetPluginsPath(const std::string& path) {

m\_PluginsPath = path;

}

// Загрузка конфигурации плагинов из стандартного файла .plugins

void sctPluginTracker::LoadDefaultPluginsConfiguration() {

if (!m\_PluginsPath.empty()) {

std::string pluginsFilePath = m\_PluginsPath + "AT66553.plugins";

LoadPluginConfigurationXML(pluginsFilePath.c\_str());

} else {

std::cerr << "Plugins path is not set." << std::endl;

}

}

// Загрузка конфигурации плагинов из XML файла

void sctPluginTracker::LoadPluginConfigurationXML(const char\* xmlFilePath) {

std::ifstream file(xmlFilePath);

if (!file.is\_open()) {

std::cerr << "Failed to open .plugins file: " << xmlFilePath << std::endl;

return;

}

std::string line;

while (std::getline(file, line)) {

std::istringstream iss(line);

std::string tag;

iss >> tag;

if (tag == "<Plugin") {

std::string name, auto\_load;

while (iss >> tag) {

if (tag.find("name=") != std::string::npos) {

name = tag.substr(6, tag.length() - 7); // Убираем кавычки

}

if (tag.find("auto\_load=") != std::string::npos) {

auto\_load = tag.substr(11, tag.length() - 12); // Убираем кавычки

}

}

if (!name.empty()) {

std::string plugin\_filename = sctLocatePluginSerial(name);

int auto\_load\_int = std::stoi(auto\_load);

PluginInfo pluginInfo;

pluginInfo.m\_FileName = plugin\_filename;

pluginInfo.m\_PluginName = name;

pluginInfo.m\_Autoload = (auto\_load\_int != 0);

m\_PluginsList.push\_back(pluginInfo);

}

}

}

file.close();

}

// Получение количества плагинов

unsigned int sctPluginTracker::GetNumberOfPlugins() {

return static\_cast<unsigned int>(m\_PluginsList.size());

}

// Регистрация доступного плагина по имени файла

unsigned int sctPluginTracker::RegisterAvailablePlugin(const std::string& filename) {

std::string defaultname = sctGetPluginNameFromFileName(filename);

auto iter = locateUsingFileName(filename);

if (iter == m\_PluginsList.end()) {

iter = locateUsingPluginName(defaultname);

}

if (iter == m\_PluginsList.end()) {

PluginInfo item;

item.m\_FileName = filename;

item.m\_PluginName = defaultname;

m\_PluginsList.push\_back(item);

return static\_cast<unsigned int>(m\_PluginsList.size() - 1);

} else {

return static\_cast<unsigned int>(iter - m\_PluginsList.begin());

}

}

// Регистрация плаг```cpp

// Регистрация плагина

void sctPluginTracker::RegisterPlugin(sctPlugin\* plugin) {

assert(plugin != nullptr);

auto iter = locateUsingPluginName(plugin->GetPluginName());

if (iter == m\_PluginsList.end()) {

PluginInfo item;

item.m\_FileName = plugin->GetPluginName() != "" ? plugin->GetFileName() : "linked-in";

item.m\_PluginName = plugin->GetPluginName();

item.m\_Plugin = plugin;

m\_PluginsList.push\_back(item);

} else {

iter->m\_Plugin = plugin;

if (plugin->GetFileName() != "") {

iter->m\_FileName = plugin->GetFileName();

}

}

}

// Получение плагина по индексу

sctPlugin\* sctPluginTracker::GetPlugin(unsigned int index) {

if (index >= this->GetNumberOfPlugins()) {

return nullptr;

}

return m\_PluginsList[index].m\_Plugin;

}

// Получение имени плагина по индексу

std::string sctPluginTracker::GetPluginName(unsigned int index) {

if (index >= this->GetNumberOfPlugins()) {

return "";

}

return m\_PluginsList[index].m\_PluginName;

}

// Получение имени файла плагина по индексу

std::string sctPluginTracker::GetPluginFileName(unsigned int index) {

if (index >= this->GetNumberOfPlugins()) {

return "";

}

return m\_PluginsList[index].m\_FileName;

}

// Проверка, загружен ли плагин по индексу

bool sctPluginTracker::GetPluginLoaded(unsigned int index) {

if (index >= this->GetNumberOfPlugins()) {

return false;

}

return m\_PluginsList[index].m\_Plugin != nullptr;

}

// Проверка, установлен ли автозагрузчик плагина по индексу

bool sctPluginTracker::GetPluginAutoLoad(unsigned int index) {

if (index >= this->GetNumberOfPlugins()) {

return false;

}

return m\_PluginsList[index].m\_Autoload;

}

// Получение списка файлов в директории

std::vector<std::string> sctPluginTracker::GetFilesInDirectory(const std::string& directory) {

std::vector<std::string> files;

for (const auto& entry : fs::directory\_iterator(directory)) {

if (entry.is\_regular\_file() && entry.path().extension() == ".plugins") {

files.push\_back(entry.path().string());

}

}

return files;

}

// Функция для выбора файла

std::string sctPluginTracker::SelectFile(const std::vector<std::string>& files) {

if (files.empty()) {

return "";

}

std::cout << "Available plugin files:" << std::endl;

for (size\_t i = 0; i < files.size(); ++i) {

std::cout << i + 1 << ": " << files[i] << std::endl;

}

size\_t choice = 0;

while (choice < 1 || choice > files.size()) {

std::cout << "Select a file by entering the corresponding number: ";

std::cin >> choice;

if (std::cin.fail() || choice < 1 || choice > files.size()) {

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cout << "Invalid choice. Please try again." << std::endl;

}

}

return files[choice - 1];

}

**Обработка открытия детали в Scientific View**

**Найти место в вашем проекте, где происходит открытие файла или детали в Scientific View. Например, это может быть в DataFileManager.**

**DataFileManager.h**

#pragma once

#include <string>

class DataFileManager {

public:

bool OnOpenDocument(std::string cFileName);

bool OnOpenDataFile(std::string cFileName);

};

**DataFileManager.cpp**

#include "DataFileManager.h"

#include "sctPluginTracker.h"

#include <iostream>

#include <algorithm>

// Функция для открытия документа

bool DataFileManager::OnOpenDocument(std::string cFileName) {

scguiFunctionsBase\* pGUIFunctionBase = gf\_GetGUIFunctionBasePointer();

std::string cFileNameLocale = cFileName;

if (pGUIFunctionBase != NULL) {

cFileNameLocale = pGUIFunctionBase->UTF8strToLocate(cFileNameLocale);

}

ClientServerOptions\* pClientServerOptions = gf\_GetClientServerOptionsPointer();

sctSCMeshDataModel\* pscMeshDataModel = gf\_GetCurSCMeshDataModel();

int ibegTime = 0;

ibegTime = GetVisTime();

CVisualOptions\* pVisualOptions = gf\_GetVisualOptionsPointer();

// Задаем путь к папке с плагинами

sctPluginTracker\* pluginTracker = sctPluginTracker::GetInstance();

pluginTracker->SetPluginsPath("D:/balka\_");

// Получение списка файлов в директории

std::vector<std::string> files = pluginTracker->GetFilesInDirectory("D:/balka\_");

// Выбор файла

std::string selectedFile = pluginTracker->SelectFile(files);

// Загружаем конфигурацию плагинов из выбранного файла

if (!selectedFile.empty()) {

pluginTracker->LoadPluginConfigurationXML(selectedFile.c\_str());

} else {

std::cerr << "No plugin files found in directory." << std::endl;

}

// Ваш текущий код...

// Пользователь открывает файл проекта

if (fileType == C\_FT\_3DRND\_Project) {

#ifndef SV\_WITHOUT\_3DRND\_MODE

CWriteReadProject\* pWRProject = gf\_GetWRProjectPointer();

flagOpen = pWRProject->ReadProject(cFileName);

scguiVisual(GUIUpdateBase\* pSCVisualGUIUpdater = gf\_GetSCVisualGUIUpdaterBasePointer();

if (pSCVisualGUIUpdater != NULL) {

pSCVisualGUIUpdater->showWizardBar();

}

#endif

}

// Дальнейшая обработка...

// ...

return true;

}

// Функция для открытия файла данных

bool DataFileManager::OnOpenDataFile(std::string cFileName) {

scguiFunctionsBase\* pGUIFunctionBase = gf\_GetGUIFunctionBasePointer();

std::string cFileNameLocale = cFileName;

if (pGUIFunctionBase != NULL) {

cFileNameLocale = pGUIFunctionBase->UTF8strToLocate(cFileNameLocale);

}

// Задаем путь к папке с плагинами

sctPluginTracker\* pluginTracker = sctPluginTracker::GetInstance();

pluginTracker->SetPluginsPath("D:/balka\_");

// Получение списка файлов в директории

std::vector<std::string> files = pluginTracker->GetFilesInDirectory("D:/balka\_");

// Выбор файла

std::string selectedFile = pluginTracker->SelectFile(files);

// Загружаем конфигурацию плагинов из выбранного файла

if (!selectedFile.empty()) {

pluginTracker->LoadPluginConfigurationXML(selectedFile.c\_str());

} else {

std::cerr << "No plugin files found in directory." << std::endl;

}

// Ваш текущий код...

// Дальнейшая обработка...

// ...

return true;

}