// Chrome Built-in AI APIs - C++ Native Messaging Host

// This demonstrates how to create a native messaging host that communicates

// with Chrome extensions to use the Built-in AI APIs

#include <iostream>

#include <string>

#include <vector>

#include <json/json.h>

#include <cstdint>

#include <cstring>

// ============================================

// NATIVE MESSAGING HELPER FUNCTIONS

// ============================================

class NativeMessagingHost {

private:

// Read message length (4 bytes)

uint32\_t readMessageLength() {

uint32\_t length = 0;

std::cin.read(reinterpret\_cast<char\*>(&length), 4);

return length;

}

// Write message length (4 bytes)

void writeMessageLength(uint32\_t length) {

std::cout.write(reinterpret\_cast<const char\*>(&length), 4);

std::cout.flush();

}

public:

// Read a message from Chrome extension

std::string readMessage() {

uint32\_t length = readMessageLength();

if (length == 0) {

return "";

}

std::vector<char> buffer(length);

std::cin.read(buffer.data(), length);

return std::string(buffer.begin(), buffer.end());

}

// Send a message to Chrome extension

void sendMessage(const std::string& message) {

uint32\_t length = static\_cast<uint32\_t>(message.length());

writeMessageLength(length);

std::cout.write(message.c\_str(), length);

std::cout.flush();

}

// Send JSON response

void sendJSONResponse(const Json::Value& response) {

Json::FastWriter writer;

std::string jsonString = writer.write(response);

sendMessage(jsonString);

}

};

// ============================================

// AI API REQUEST HANDLERS

// ============================================

class AIAPIHandler {

private:

NativeMessagingHost& host;

public:

AIAPIHandler(NativeMessagingHost& h) : host(h) {}

// Handle Prompt API request

void handlePromptRequest(const Json::Value& request) {

std::string text = request["text"].asString();

std::string systemPrompt = request.get("systemPrompt", "").asString();

// Create response for Chrome extension to process

Json::Value response;

response["type"] = "promptRequest";

response["action"] = "generate";

response["text"] = text;

response["systemPrompt"] = systemPrompt;

response["temperature"] = request.get("temperature", 0.7).asDouble();

response["topK"] = request.get("topK", 3).asInt();

host.sendJSONResponse(response);

}

// Handle Proofreader API request

void handleProofreaderRequest(const Json::Value& request) {

std::string text = request["text"].asString();

Json::Value response;

response["type"] = "proofreadRequest";

response["action"] = "proofread";

response["text"] = text;

host.sendJSONResponse(response);

}

// Handle Summarizer API request

void handleSummarizerRequest(const Json::Value& request) {

std::string text = request["text"].asString();

std::string summaryType = request.get("summaryType", "key-points").asString();

Json::Value response;

response["type"] = "summarizeRequest";

response["action"] = "summarize";

response["text"] = text;

response["summaryType"] = summaryType;

response["length"] = request.get("length", "medium").asString();

host.sendJSONResponse(response);

}

// Handle Translator API request

void handleTranslatorRequest(const Json::Value& request) {

std::string text = request["text"].asString();

std::string sourceLang = request["sourceLanguage"].asString();

std::string targetLang = request["targetLanguage"].asString();

Json::Value response;

response["type"] = "translateRequest";

response["action"] = "translate";

response["text"] = text;

response["sourceLanguage"] = sourceLang;

response["targetLanguage"] = targetLang;

host.sendJSONResponse(response);

}

// Handle Writer API request

void handleWriterRequest(const Json::Value& request) {

std::string context = request["context"].asString();

Json::Value response;

response["type"] = "writeRequest";

response["action"] = "write";

response["context"] = context;

response["tone"] = request.get("tone", "neutral").asString();

response["length"] = request.get("length", "medium").asString();

host.sendJSONResponse(response);

}

// Handle Rewriter API request

void handleRewriterRequest(const Json::Value& request) {

std::string text = request["text"].asString();

Json::Value response;

response["type"] = "rewriteRequest";

response["action"] = "rewrite";

response["text"] = text;

response["tone"] = request.get("tone", "more-professional").asString();

response["length"] = request.get("length", "as-is").asString();

host.sendJSONResponse(response);

}

};

// ============================================

// MESSAGE PROCESSOR

// ============================================

class MessageProcessor {

private:

NativeMessagingHost& host;

AIAPIHandler apiHandler;

public:

MessageProcessor(NativeMessagingHost& h) : host(h), apiHandler(h) {}

void processMessage(const std::string& messageStr) {

Json::Value message;

Json::Reader reader;

if (!reader.parse(messageStr, message)) {

sendError("Invalid JSON format");

return;

}

std::string apiType = message.get("api", "").asString();

if (apiType == "prompt") {

apiHandler.handlePromptRequest(message);

} else if (apiType == "proofreader") {

apiHandler.handleProofreaderRequest(message);

} else if (apiType == "summarizer") {

apiHandler.handleSummarizerRequest(message);

} else if (apiType == "translator") {

apiHandler.handleTranslatorRequest(message);

} else if (apiType == "writer") {

apiHandler.handleWriterRequest(message);

} else if (apiType == "rewriter") {

apiHandler.handleRewriterRequest(message);

} else {

sendError("Unknown API type: " + apiType);

}

}

void sendError(const std::string& error) {

Json::Value response;

response["type"] = "error";

response["message"] = error;

host.sendJSONResponse(response);

}

};

// ============================================

// SMART TEXT ANALYZER (Example Application)

// ============================================

class SmartTextAnalyzer {

private:

NativeMessagingHost& host;

public:

SmartTextAnalyzer(NativeMessagingHost& h) : host(h) {}

void analyzeText(const std::string& text) {

// Request multiple AI operations

Json::Value analysis;

analysis["type"] = "batchAnalysis";

analysis["text"] = text;

// Request operations

Json::Value operations(Json::array