

Akcja Algorytm C# — Instrukcja użytkownika

SPIS TREŚCI

Przeczytasz w 23 min.

- [Wprowadzenie](#)
 - [Co można zrobić za pomocą Algorytmu C#?](#)
 - [Wymagane uprawnienia](#)
- [Tworzenie akcji Algorytm C#](#)
- [Edytor kodu](#)
 - [Opcje konfiguracji](#)
 - [Zasady pisania skryptów](#)
- [Dostępne zmienne kontekstowe](#)
 - [Deserializacja modelu](#)
- [Wykonywanie zapytań SQL](#)
 - [Pobranie wielu wierszy](#)
 - [Pobranie pojedynczej wartości](#)
 - [Wykonanie INSERT/UPDATE/DELETE](#)
 - [Użycie innego źródła danych](#)
- [Uruchamianie innych akcji](#)
 - [Wysłanie emaila](#)
 - [Zapis rekordu dokumentu](#)
 - [Zmiana statusu dokumentu](#)
 - [Uruchomienie innej akcji Algorytm C#](#)
 - [Uruchomienie akcji SQL](#)
- [Wywoływanie usług REST i SOAP](#)
 - [Wywołanie REST API](#)
 - [Typy uwierzytelniania \(AuthType\)](#)
 - [Wywołanie SOAP](#)
 - [Właściwości odpowiedzi \(IResponse\)](#)
- [Praca z załącznikami](#)
 - [Pobranie listy załączników](#)

- [Pobranie zawartości załącznika](#)
- [Wgranie załącznika](#)
- [Usunięcie załącznika](#)
- [Zmiana opisu załącznika](#)
- [Logowanie](#)
- [Obsługa błędów](#)
 - [Rzucanie wyjątków z komunikatem tłumaczącym](#)
 - [Rzucanie wyjątku z wynikiem walidacji](#)
 - [Rzucanie z formatowanym HTML](#)
 - [Sprawdzanie wyników akcji](#)
- [Sekrety i konfiguracja](#)
 - [Odczyt sekretu tenanta](#)
 - [Odczyt sekretu środowiska](#)
 - [Czyszczenie cache'a](#)
- [Dodatkowe referencje do bibliotek](#)
 - [Assembly References](#)
 - [Library Packages](#)
- [Kompilacja i testowanie](#)
 - [Kompilacja](#)
 - [Diagnostyka](#)
 - [Wynik wykonania](#)
- [Przykłady praktyczne](#)
 - [Przykład 1: Masowa zmiana statusu z powiadomieniem email](#)
 - [Przykład 2: Integracja z zewnętrznym API](#)
 - [Przykład 3: Generowanie i dołączanie pliku](#)
- [FAQ i najczęstsze problemy](#)
 - [P: Skrypt się kompiluje, ale nie wykonuje się poprawnie](#)
 - [P: Otrzymuję błąd "Wykonywana akcja nie jest typu algorytm C#"](#)
 - [P: Zapytanie SQL zwraca null](#)
 - [P: Jak odwołać się do wartości z formularza?](#)
 - [P: Jak poznać TransitionInternalId?](#)
 - [P: Czy mogę definiować własne metody?](#)
 - [P: Jak bezpiecznie przechowywać dane uwierzytelniające?](#)
 - [P: Skrypt działa wolno](#)

- [P: Jak zwrócić wartość ze skryptu?](#)

WPROWADZENIE

Akcja Algorytm C# (CSharpScriptAction) to mechanizm systemu nAxiom umożliwiający pisanie i wykonywanie kodu C# bezpośrednio w aplikacji. Dzięki temu administratorzy i programiści mogą tworzyć zaawansowaną logikę biznesową, automatyzacje i integracje bez konieczności modyfikacji kodu źródłowego systemu.

Co można zrobić za pomocą Algorytmu C#?

- Wykonywać zapytania SQL do bazy danych
- Wywoływać inne akcje systemu (email, zmiana statusu, zapis rekordu)
- Komunikować się z zewnętrznymi serwisami REST i SOAP
- Operować na załącznikach dokumentów
- Logować informacje diagnostyczne
- Tworzyć złożone scenariusze automatyzacji

Wymagane uprawnienia

Do uruchomienia akcji wymagane jest uprawnienie systemowe **AllAppUsage**.

TWORZENIE AKCJI ALGORYTM C#

- 1 Przejdź do modułu **Administracja** → **Akcje**
- 2 Kliknij **Dodaj nową akcję**
- 3 Wypełnij wymagane pola (nazwa, opis)
- 4 Jako typ akcji wybierz **Algorytm C#**
- 5 W edytorze kodu wpisz swój skrypt C#
- 6 Kliknij **Kompiluj** aby sprawdzić poprawność kodu
- 7 Po pozytywnej kompilacji kliknij **Zapisz**

Uwaga: Zapisanie akcji wymaga pomyślnej kompilacji skryptu. Akcja z błędami kompilacji nie zostanie zapisana.

EDYTOR KODU

Po wybraniu typu akcji Algorytm C# system wyświetli edytor kodu z domyślnymi instrukcjami

```
using:
```

```
using System;
using System.Linq;
using System.Text;
using CSharpScript.Attachments;
using CSharpScript.ActionRunner;
using CSharpScript.SqlRunner;
using CSharpScript.WebServiceRunner;
using CSharpScript.WebServiceRunner.Models;
using Microsoft.Extensions.Logging;
```

Opcje konfiguracji

Opcja	Opis
Debug Information	Checkbox - włącza dodatkowe informacje debugowania
Assembly References	Lista dodatkowych bibliotek DLL do załadowania
Library Packages	Paczki bibliotek wgrane do systemu (opcjonalne)

Zasady pisania skryptów

- Kod jest pisany jako **skrypt** — nie potrzeba klasy ani metody `Main`
- Kod jest **async** — można używać `await` bezpośrednio
- Zmienne systemowe (np. `RecordId`, `Model`) są dostępne bezpośrednio
- Na końcu skryptu można definiować prywatne metody pomocnicze
- Ostatnia wartość wyrażenia w skrypcie staje się wartością zwracaną

DOSTĘPNE ZMIENNE KONTEKSTOWE

W trakcie wykonywania skryptu dostępne są następujące zmienne z kontekstem aktualnego wywołania:

Zmienna	Typ	Opis
<code>RecordId</code>	<code>string</code>	ID rekordu dokumentu, na którym akcja jest uruchomiona
<code>Model</code>	<code>string</code>	Dane formularza w formacie JSON
<code>UserId</code>	<code>Guid?</code>	ID zalogowanego użytkownika
<code>GridId</code>	<code>int?</code>	ID gridy, z którego uruchomiono akcję

Zmienna	Typ	Opis
FormDefinitionId	int?	ID definicji formularza
BusinessDocumentId	int?	ID definicji dokumentu biznesowego
TenantId	string	Identyfikator tenanta
TenantBaseUrl	string	Bazowy URL aplikacji (np. <code>https://mojafirma.naxiom.com</code>)

Deserializacja modelu

```
var json = Newtonsoft.Json.JsonConvert.DeserializeObject<Dictionary<string, object>>(Model);
// json["NazwaPola"] – dostęp do wartości pola z formularza
```

WYKONYWANIE ZAPYTAŃ SQL

Obiekt `SqlHelper` umożliwia wykonywanie zapytań do bazy danych.

Pobranie wielu wierszy

```
var wyniki = await SqlHelper.ExecuteSqlQueryRowsToDictionary(
    "SELECT Id, Name, Email FROM dbo.Klienci WHERE Status = @Status",
    new Dictionary<string, object> { { "@Status", "Aktywny" } });

foreach (var wiersz in wyniki)
{
    Logger.LogInformation($"Klient: {wiersz["Name"]}");
}
```

Pobranie pojedynczej wartości

```
var liczbaDokumentow = (int)await SqlHelper.ExecuteSqlScalar(
    "SELECT COUNT(*) FROM dbo.Dokumenty WHERE BusinessDocumentId = @BdId",
    new Dictionary<string, object> { { "@BdId", BusinessDocumentId } });
```

Wykonanie INSERT/UPDATE/DELETE

```
var sukces = await SqlHelper.ExecuteSqlNonQuery(
    "UPDATE dbo.Klienci SET Status = @NowyStatus WHERE Id = @Id",
    new Dictionary<string, object>
    {
        { "@NowyStatus", "Zamknięty" },
        { "@Id", RecordId }
    });
```

Użycie innego źródła danych

Wszystkie metody SQL przyjmują opcjonalny parametr `dataSourceId` pozwalający na zapytanie do innej bazy danych:

```
var wyniki = await SqlHelper.ExecuteSqlQueryRowsToDictionary(
    "SELECT * FROM dbo.Faktury",
    null,
    dataSourceId: 5); // ID źródła danych skonfigurowanego w systemie
```

Wskazówka: Zawsze używaj parametrów (`@Param`) zamiast wklejania wartości bezpośrednio do zapytania — zapobiega to atakom SQL Injection.

URUCHAMIANIE INNYCH AKCJI

Wysłanie emaila

```
using CSharpScript.ActionRunner;
using Newtonsoft.Json;

var parametryMail = new Dictionary<string, object>
{
    { "RecordId", RecordId },
    { "Model", JsonConvert.SerializeObject(new { Temat = "Potwierdzenie", Email = "jan@firma.pl" }) },
    { "UserId", UserId },
    { "BusinessDocumentId", BusinessDocumentId }
};

var wynik = await EmailAction.Instance.RunActionAsync(
    123, // ID akcji email w systemie
    JsonConvert.SerializeObject(parametryMail));
```

```
if (!wynik.IsValid)
{
    throw new Exception($"Błąd wysyłki: {JsonConvert.SerializeObject(wynik.ResponseMessages)}");
}
```

Zapis rekordu dokumentu

```
using CSharpScript.ActionRunner;

// Tworzenie nowego rekordu (RecordId = "0")
var request = new SaveDocumentRequest()
{
    RecordId = "0",
    BusinessDocumentId = BusinessDocumentId,
    Model = new Dictionary<string, object>()
    {
        { "Email", "jan.nowak@firma.pl" },
        { "Nazwa", "Jan Nowak" },
        { "DataUtworzenia", DateTime.Now.ToString("yyyy-MM-dd") }
    }
};

var wynik = await SaveRecordAction.Instance.RunActionAsync(request);

if (!wynik.IsValid)
{
    throw new Shared.Exceptions.TranslatedException(
        "Tworzenie rekordu nie powiodło się",
        wynik.ValidationResult);
}

var noweId = wynik.Result; // ID utworzonego rekordu
```

Zmiana statusu dokumentu

```
using CSharpScript.ActionRunner;
using Newtonsoft.Json;

var request = new ChangeStatusRequest()
```

```

{
    RecordId = RecordId,
    BusinessDocumentId = BusinessDocumentId,
    TransitionInternalId = "c70a1fd2-32b2-7d0b-c470-d42eba506425", // GUID przejścia
    Model = JsonConvert.DeserializeObject<Dictionary<string, object>>(Model)
};

var wynik = await ChangeStatusAction.Instance.RunActionAsync(request);

if (!wynik.IsValid)
{
    throw new Shared.Exceptions.TranslatedException(
        "Zmiana statusu nie powiodła się",
        wynik.ValidationResult);
}

```

Wskazówka: `TransitionInternalId` to wewnętrzny identyfikator przejścia (GUID). Możesz go znaleźć w tabeli `core.BusinessTransitions`.

Uruchomienie innej akcji Algorytm C#

```

var wynik = await CSharpScriptAction.Instance.RunActionAsync(
    456, // ID innej akcji C#
    JsonConvert.SerializeObject(new Dictionary<string, object>
    {
        { "RecordId", RecordId },
        { "Model", Model },
        { "BusinessDocumentId", BusinessDocumentId }
    }));

```

Uruchomienie akcji SQL

```

var wynik = await SqlAction.Instance.RunActionAsync(
    789, // ID akcji SQL
    JsonConvert.SerializeObject(new Dictionary<string, object>
    {
        { "RecordId", RecordId },
        { "Model", Model }
    }));

```

WYWOŁYWANIE USŁUG REST I SOAP

Wywołanie REST API

```
using CSharpScript.WebServiceRunner.Models;
using System.Net.Http;

var request = new RestRequestModel()
{
    Method = HttpMethod.Post,
    Url = "https://api.zewnetrzny.pl/zamowienia",
    Headers = new Dictionary<string, string>
    {
        { "Content-Type", "application/json" }
    },
    Body = Newtonsoft.Json.JsonConvert.SerializeObject(new
    {
        numerZamowienia = RecordId,
        kwota = 100.50
    }),
    AuthType = ApiConnector.Enums.AuthType.Base64,
    UserName = "mojeKonto",
    Password = "mojeHaslo"
};

var response = await RestManager.Instance.ExecuteRequestAsync(request);

if (response.IsSuccess)
{
    Logger.LogInformation($"Sukces: {response.RawResult}");
}
else
{
    throw new Exception($"Błąd: {response.ResponseStatusCode} - {response.RawResult}");
}
```

Typy uwierzytelniania (AuthType)

Wartość	Opis	Wymagane pola
<code>AuthType.NoAuth</code>	Bez uwierzytelniania	—
<code>AuthType.Base64</code>	HTTP Basic Auth	UserName, Password
<code>AuthType.JWT</code>	Token JWT	UserName, Password, AuthUrl
<code>AuthType.OAuth2Password</code>	OAuth2 Password Grant	UserName, Password, AuthUrl, ClientId, ClientSecret, Scope
<code>AuthType.OAuth2ClientCredentials</code>	OAuth2 Client Credentials	AuthUrl, ClientId, ClientSecret, Scope

Wywołanie SOAP

```
using CSharpScript.WebServiceRunner.Models;

var soapBody = @"<?xml version='1.0' encoding='utf-8'?>
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>
  <soap:Body>
    <GetOrderStatus xmlns='http://example.com/orders'>
      <OrderId>12345</OrderId>
    </GetOrderStatus>
  </soap:Body>
</soap:Envelope>";

var request = new SoapRequestModel()
{
    Url = "https://example.com/orders/service",
    Action = "http://example.com/orders/GetOrderStatus",
    Body = soapBody,
    AuthType = ApiConnector.Enums.AuthType.Base64,
    UserName = "soapUser",
    Password = "soapPassword"
};

var response = await SoapManager.Instance.ExecuteRequestAsync(request);
```

Właściwości odpowiedzi (IResponse)

Właściwość	Typ	Opis
IsSuccess	bool	Czy żądanie zakończyło się sukcesem
RawResult	string	Surowa treść odpowiedzi
ResponseStatusCode	HttpStatusCode	Kod statusu HTTP (200, 404, 500 itp.)
ResponseHeaders	Dictionary	Nagłówki odpowiedzi

PRACA Z ZAŁĄCZNIKAMI

Obiekt `DocumentAttachmentsHelper` umożliwia operacje CRUD na załącznikach dokumentów biznesowych.

Pobranie listy załączników

```
using CSharpScript.Attachments;

var attachments = await DocumentAttachmentsHelper.GetAttachments(new GetAttachmentsRequest
{
    BusinessDocumentId = BusinessDocumentId.Value,
    RecordId = RecordId,
    AttachmentCategoryId = null // opcjonalnie: filtruj po kategorii
});

foreach (var att in attachments.Result)
{
    Logger.LogInformation($"Załącznik: {att.FileName} ({att.FileSize} bajtów)");
}
```

Pobranie zawartości załącznika

```
var attachment = await DocumentAttachmentsHelper.GetAttachmentWithFileContent(attachmentId);
byte[] fileContent = attachment.Result.FileContent;
string fileName = attachment.Result.FileName;
```

Wgranie załącznika

```

var result = await DocumentAttachmentsHelper.UploadAttachment(new UploadAttachmentRequest
{
    RecordId = RecordId,
    BusinessDocumentId = BusinessDocumentId.Value,
    FileName = "raport.pdf",
    Description = "Wygenerowany raport",
    FileContent = pdfBytes,           // byte[]
    Overwrite = false,                // true = nadpisz jeśli istnieje
    AttachmentCategoryId = null      // opcjonalna kategoria
});

```

Usunięcie załącznika

```

await DocumentAttachmentsHelper.RemoveAttachment(attachmentId);           // usunięcie z wszystkimi wersjami
await DocumentAttachmentsHelper.RemoveAttachmentVersion(attachmentId); // usunięcie tylko wybranej wersji

```

Zmiana opisu załącznika

```

await DocumentAttachmentsHelper.SetAttachmentDescription(new SetAttachmentDescriptionRequest
{
    AttachmentId = 123,
    Description = "Zaktualizowany opis"
});

```

LOGOWANIE

Obiekt `Logger` umożliwia zapisywanie informacji diagnostycznych:

```

Logger.LogDebug("Szczegółowa informacja do debugowania");
Logger.LogInformation("Informacja: przetworzono {Count} rekordów", count);
Logger.LogWarning("Ostrzeżenie: brak danych dla rekordu {Id}", RecordId);
Logger.LogError("Błąd: {Message}", ex.Message);
Logger.LogCritical("Krytyczny błąd: {Error}", error);

```

Wskazówka: Logi są zapisywane zgodnie z konfiguracją NLog systemu nAxiom. Używaj logowania, aby ułatwić diagnozowanie problemów w skryptach.

OBSŁUGA BŁĘDÓW

Rzucanie wyjątków z komunikatem tłumaczącym

```
using Shared.Exceptions;

throw new TranslatedException("Operacja nie powiodła się. Skontaktuj się z administratorem.");
```

Rzucanie wyjątku z wynikiem walidacji

```
throw new TranslatedException("Walidacja nie powiodła się", wynik.ValidationResult);
```

Rzucanie z formatowanym HTML

```
throw new Exception(
    $"Błąd: {response.RawResult}<br>" +
    $"Skontaktuj się z działem IT.<br>" +
    $"<a href='mailto:it@firma.pl'>Zgłoś błąd</a>");
```

Sprawdzanie wyników akcji

```
var wynik = await EmailAction.Instance.RunActionAsync(id, model);

if (!wynik.IsValid)
{
    // wynik.ResponseMessages – lista komunikatów błędów
    // wynik.ValidationResult – szczegóły walidacji
    var komunikaty = string.Join(", ", wynik.ResponseMessages.Select(m => m.Message));
    throw new Exception($"Akcja nie powiodła się: {komunikaty}");
}
```

SEKRETY I KONFIGURACJA

Odczyt sekretu tenanta

```
byte[] certyfikat = await TenantsAdmin.CurrentTenant.Secrets.GetSecretFile("MojCertyfikatSSL");
```

Odczyt sekretu środowiska

```
byte[] klucz = await TenantsAdmin.Environment.Secrets.GetSecretFile("ApiKeyProduction");
```

Czyszczenie cache'a

```
ClearCache(); // Czyści globalny cache aplikacji
```

Uwaga: `ClearCache()` czyści cache dla **wszystkich tenantów**. Używaj ostrożnie.

DODATKOWE REFERENCJE DO BIBLIOTEK

Assembly References

W konfiguracji akcji można dodać dodatkowe DLL, które będą załadowane podczas kompilacji:

- `System.Data.SqlClient.dll` — bezpośredni dostęp do SQL Server
- `Dapper.dll` — mikro-ORM
- `Newtonsoft.Json.dll` — już domyślnie dostępne

Library Packages

System nAxiom pozwala na wgrywanie własnych paczek DLL (Library Packages) przez panel administracyjny. Po wgraniu paczki, jej ID (GUID) można wybrać w konfiguracji akcji.

KOMPILACJA I TESTOWANIE

Kompilacja

- Kliknij przycisk **Kompiluj** w edytorze aby zweryfikować poprawność kodu
- System zwraca listę błędów kompilacji z numerem linii i opisem
- **Zielony komunikat** = kompilacja poprawna
- **Czerwony komunikat** = błędy kompilacji

Diagnostyka

W przypadku błędów wyświetlane są informacje:

- **Type:** Typ diagnostyki (Error, Warning, Info)
- **Code:** Kod błędu (np. CS0103)
- **Message:** Opis błędu
- **Line/Character:** Pozycja błędu w kodzie

Wynik wykonania

Po uruchomieniu akcji system zwraca:

- **ExecutedSuccessfully:** Czy skrypt wykonał się poprawnie
- **CompiledSuccessfully:** Czy kompilacja się powiodła
- **ReturnValue:** Wartość zwrócona przez skrypt (JSON)
- **Diagnostics:** Lista błędów/ostrzeżeń

PRZYKŁADY PRAKTYCZNE

Przykład 1: Masowa zmiana statusu z powiadomieniem email

Scenariusz: Zmień status wszystkich ticketów z przekroczonym terminem i wyślij powiadomienie.

```
using System;
using System.Linq;
using System.Collections.Generic;
using Newtonsoft.Json;
using CSharpScript.ActionRunner;
using CSharpScript.SqlRunner;

// Pobranie konfiguracji
int ticketBdId = (int)await SqlHelper.ExecuteSqlScalar(
    "SELECT Id FROM core.BusinessDocuments WHERE Code='Ticket'");

// Pobranie ticketów do zmiany
var tickety = await SqlHelper.ExecuteSqlQueryRowsToDictionary(
    @"SELECT Id FROM dbo.Ticket
    WHERE StatusName <> 'Zamknięty'
    AND DATEDIFF(DAY, GETDATE(), TerminRealizacji) < 0");

Logger.LogInformation($"Znaleziono {tickety.Count} przeterminowanych ticketów");

foreach (var ticket in tickety)
{
    // Zmiana statusu
    var zmiana = new ChangeStatusRequest()
    {
        RecordId = ticket["Id"].ToString(),
        BusinessDocumentId = ticketBdId,
        TransitionInternalId = "guid-przejscia-na-przeterminowany",
    };
}
```

```

        Model = new Dictionary<string, object> { { "_", "_" } }
    };

    var wynikZmiany = await ChangeStatusAction.Instance.RunActionAsync(zmiana);

    if (!wynikZmiany.IsValid)
    {
        Logger.LogWarning($"Nie udało się zmienić statusu ticketa {ticket["Id"]}");
        continue;
    }

    // Wysłanie powiadomienia email
    var mailModel = new Dictionary<string, object>
    {
        { "RecordId", ticket["Id"] },
        { "Model", JsonConvert.SerializeObject(new { TicketId = ticket["Id"] }) },
        { "UserId", UserId },
        { "BusinessDocumentId", ticketBdId }
    };

    await EmailAction.Instance.RunActionAsync(1, JsonConvert.SerializeObject(mailModel));
}

Logger.LogInformation("Przetwarzanie zakończone");

```

Przykład 2: Integracja z zewnętrznym API

Scenariusz: Pobierz dane z zewnętrznego API i zapisz w systemie.

```

using System;
using System.Net.Http;
using CSharpScript.WebServiceRunner.Models;
using CSharpScript.ActionRunner;
using Newtonsoft.Json;

// Pobranie danych z API
var request = new RestRequestModel()
{
    Method = HttpMethod.Get,
    Url = "https://api.dostawca.pl/produkty?category=nowe",
    AuthType = ApiConnector.Enums.AuthType.OAuth2ClientCredentials,
}

```

```

    AuthUrl = "https://api.dostawca.pl/oauth/token",
    ClientId = "moj-client-id",
    ClientSecret = "moj-secret",
    Scope = "products.read"
};

var response = await RestManager.Instance.ExecuteRequestAsync(request);

if (!response.IsSuccess)
{
    throw new Exception($"Błąd pobierania produktów: {response.RawResult}");
}

var produkty = JsonConvert.DeserializeObject<List<dynamic>>(response.RawResult);

Logger.LogInformation($"Pobrano {produkty.Count} produktów");

// Zapisanie produktów jako rekordów dokumentu biznesowego
foreach (var produkt in produkty)
{
    var saveRequest = new SaveDocumentRequest()
    {
        RecordId = "0", // nowy rekord
        BusinessDocumentId = BusinessDocumentId,
        Model = new Dictionary<string, object>()
        {
            { "NazwaProduktu", (string)produkt.name },
            { "Cena", (decimal)produkt.price },
            { "KodDostawcy", (string)produkt.sku }
        }
    };

    var wynik = await SaveRecordAction.Instance.RunActionAsync(saveRequest);

    if (!wynik.IsValid)
    {
        Logger.LogWarning($"Nie udało się zapisać produktu: {produkt.name}");
    }
}

```

Przykład 3: Generowanie i dołączanie pliku

Scenariusz: Wygeneruj plik CSV i dołącz jako załącznik.

```
using System;
using System.Text;
using System.Linq;
using CSharpScript.Attachments;

// Pobranie danych
var dane = await SqlHelper.ExecuteSqlQueryRowsToDictionary(
    "SELECT Nazwa, Email, DataRejestracji FROM dbo.Klienci WHERE Status = 'Aktywny'");

// Generowanie CSV
var csv = new StringBuilder();
csv.AppendLine("Nazwa;Email;Data Rejestracji");

foreach (var wiersz in dane)
{
    csv.AppendLine($"{wiersz["Nazwa"]};{wiersz["Email"]};{wiersz["DataRejestracji"]}");
}

// Dołączenie jako załącznik
var wynik = await DocumentAttachmentsHelper.UploadAttachment(new UploadAttachmentRequest
{
    RecordId = RecordId,
    BusinessDocumentId = BusinessDocumentId.Value,
    FileName = $"export_klientow_{DateTime.Now:yyyyMMdd}.csv",
    Description = "Automatyczny eksport klientów",
    FileContent = Encoding.UTF8.GetBytes(csv.ToString()),
    Overwrite = true
});

if (wynik.IsValid)
{
    Logger.LogInformation($"Plik CSV został dołączony: {wynik.Result.FileName}");
}
```

FAQ I NAJCZĘSTSZE PROBLEMY

P: Skrypt się kompiluje, ale nie wykonuje się poprawnie

○: Sprawdź logi systemu. Użyj `Logger.LogInformation()` do diagnostyki. Sprawdź czy zmienne kontekstowe (np. `RecordId`, `BusinessDocumentId`) mają oczekiwane wartości.

P: Otrzymuję błąd "Wykonywana akcja nie jest typu algorytm C#"

○: Upewnij się, że akcja o podanym ID ma typ **CSharpScript** (Algorytm C#). Sprawdź konfigurację akcji.

P: Zapytanie SQL zwraca null

○: Sprawdź zapytanie bezpośrednio w bazie danych. Upewnij się, że parametry (`@Param`) mają poprawne wartości. Przy `ExecuteSqlScalar` sprawdź rzutowanie typu.

P: Jak odwołać się do wartości z formularza?

○: Dane formularza są w zmiennej `Model` jako JSON. Zdeserializuj je:

```
var json = JsonConvert.DeserializeObject<Dictionary<string, object>>(Model);  
var wartoscPola = json["NazwaPola"];
```

P: Jak poznać `TransitionInternalId`?

○: Wykonaj zapytanie:

```
var transitions = await SqlHelper.ExecuteSqlQueryRowsToDictionary(  
    "SELECT Id, InternalId, Name FROM core.BusinessTransitions WHERE BusinessDocumentId = @BdId",  
    new Dictionary<string, object> { { "@BdId", BusinessDocumentId } });
```

P: Czy mogę definiować własne metody?

○: Tak, na końcu skryptu możesz definiować lokalne metody:

```
// Główna logika  
var wynik = await MojaMetoda(RecordId);  
  
// Definicja metody na końcu skryptu  
private async Task<string> MojaMetoda(string id)  
{  
    return await SqlHelper.ExecuteSqlScalar($"SELECT Name FROM dbo.Table WHERE Id = {id}") as string;  
}
```

P: Jak bezpiecznie przechowywać dane uwierzytelniające?

O: Używaj systemu sekretów:

```
byte[] secretBytes = await TenantsAdmin.CurrentTenant.Secrets.GetSecretFile("APICredentials");  
string credentials = System.Text.Encoding.UTF8.GetString(secretBytes);
```

P: Skrypt działa wolno

O:

- Skompilowane skrypty są cache'owane — pierwsze uruchomienie jest wolniejsze
- Optymalizuj zapytania SQL
- Unikaj wielu sekwencyjnych wywołań HTTP — rozważ przetwarzanie wsadowe
- Włącz opcję **Debug Information** tylko do testów

P: Jak zwrócić wartość ze skryptu?

O: Ostatnia ewaluowana linia jest wartością zwracaną:

```
// ... Logika ...  
"Operacja zakończona pomyślnie" // ta wartość zostanie zwrócona
```