DynaFetch Troubleshooting Guide

Solutions for common issues when using DynaFetch in Dynamo.

Installation Issues

DynaFetch Package Not Found

Problem: Can't find DynaFetch in Dynamo Package Manager

Solutions:

1. **Check Dynamo Version**: DynaFetch requires Dynamo 3.0 or later
2. **Update Package List**: Go to Packages → Search for a Package → Refresh list
3. **Manual Installation**: Download .dll from GitHub releases and place in Dynamo packages folder
4. **Check Internet**: Package Manager requires internet connectivity

Verification:

```
// After installation, these nodes should be available:
ClientNodes.Create()
ExecuteNodes.GET()
JsonNodes.ToDictionary()
```

Package Loading Errors

Problem: DynaFetch installed but nodes don't appear

Solutions:

1. **Restart Dynamo**: Close and reopen Dynamo completely
2. **Check Dependencies**: Ensure .NET 8 runtime is installed
3. **Check Conflicts**: Disable other HTTP/REST packages temporarily
4. **Clear Cache**: Delete Dynamo's node cache and restart

File Locations:

- **Windows**: `%APPDATA%\Dynamo\Dynamo Core\2.x\packages`
- **Dynamo Cache**: `%APPDATA%\Dynamo\Dynamo Core\2.x\nodeCache`

Connection Issues

Cannot Connect to API

Error: "Unable to connect to the remote server"

Diagnostic Steps:

1. **Test URL in Browser**: Verify URL is accessible
2. **Check Network**: Ensure internet connectivity
3. **Verify HTTPS**: Most modern APIs require HTTPS
4. **Check Firewall**: Corporate firewalls may block API access

Solutions:

```
// Test basic connectivity
client = ClientNodes.Create()
response = ExecuteNodes.GET(client, "https://httpbin.org/get")
if response.IsSuccessful:
    // Network is working
    content = JsonNodes.GetContent(response)
```

Timeout Errors

Error: "The operation has timed out"

Solutions:

```
// Increase timeout for slow APIs
client = ClientNodes.Create()
ClientNodes.SetTimeout(client, 300)  // 5 minutes

// Or use shorter timeout for testing
ClientNodes.SetTimeout(client, 10)   // 10 seconds
```

Typical Timeout Values:

- **Fast APIs**: 15-30 seconds
- **Standard APIs**: 60-120 seconds
- **Large data/files**: 300+ seconds

DNS Resolution Errors

Error: "No such host is known"

Solutions:

1. **Check URL Spelling**: Verify domain name is correct
2. **Test DNS**: Use command prompt: `nslookup api.example.com`
3. **Try Different DNS**: Switch to public DNS (8.8.8.8, 1.1.1.1)
4. **Check VPN**: VPN may affect DNS resolution

Authentication Issues

401 Unauthorized Errors

Error: HTTP 401 status code

Diagnostic Steps:

```
// Check if headers are being sent
client = ClientNodes.Create()
ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer " + token)
headers = ClientNodes.GetDefaultHeaders(client)
// Verify "Authorization" key exists with correct value
```

Common Causes & Solutions:

Expired Token:

```
// Check token expiration
if token_expiry_time < current_time:
    // Refresh token before use
    new_token = get_fresh_token()
    ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer " + new_token)
```

Wrong Header Format:

```
// Correct Bearer token format
ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer " + token)

// Not: "Bearer: token" or just token
```

Wrong Header Name:

```
// Check API docs for correct header name
ClientNodes.AddDefaultHeader(client, "X-API-Key", api_key)        // Common
ClientNodes.AddDefaultHeader(client, "Api-Token", api_key)        // Alternative
ClientNodes.AddDefaultHeader(client, "X-Auth-Token", api_key)     // Alternative
```

403 Forbidden Errors

Error: HTTP 403 status code

Causes & Solutions:

- **Insufficient Permissions**: API key has wrong scope/permissions
- **Rate Limiting**: Too many requests (try with delays)
- **IP Restrictions**: API may be IP-whitelisted
- **Resource Permissions**: User doesn't have access to specific resource

Debugging:

```
response = ExecuteNodes.GET(client, url)
if response.StatusCode == 403:
    error_content = JsonNodes.GetContent(response)
    // Check error message for specific cause
```

API Key Issues

Problem: API key not working despite correct format

Solutions:

1. **Verify Key**: Test with API provider's test tool
2. **Check Scope**: Ensure key has required permissions
3. **Check Expiration**: Some API keys expire
4. **Environment**: Make sure using correct key (dev vs prod)

Testing API Keys:

```
// Test with a simple endpoint first
client = ClientNodes.Create()
ClientNodes.AddDefaultHeader(client, "X-API-Key", api_key)
response = ExecuteNodes.GET(client, "https://api.example.com/test")
// Should return 200 if key is valid
```

JWT Authentication Issues

JWT Assertion Generation Errors

Error: "Invalid private key format" or "Key parsing failed"

Common Causes & Solutions:

Wrong Key Format:

```
```

```
// DynaFetch supports PKCS#1 and PKCS#8 formats
// PKCS#1 starts with: -----BEGIN RSA PRIVATE KEY-----
// PKCS#8 starts with: -----BEGIN PRIVATE KEY-----

// Both formats work - verify your key has proper headers/footers
```

Missing Newlines in Key:

```
// Private keys must preserve line breaks
// WRONG: "-----BEGIN RSA PRIVATE KEY-----MIIEpAIBAAKC..."
// RIGHT: "-----BEGIN RSA PRIVATE KEY-----\nMIIEpAIB...\n-----END RSA PRIVATE KEY---
--"

// Load from file to preserve formatting
privateKeyPem = File.ReadAllText("C:\\keys\\private_key.pem")
```

Wrong Key Type:

```
// Must be RSA private key, not:
// - Public key
// - Certificate
// - Other key types (EC, DSA)

// Verify with: openssl rsa -in key.pem -text -noout
```

JWT Token Exchange Errors

Error: 400 or 401 when exchanging JWT for access token

Diagnostic Steps:

```
// Generate JWT
jwt = ClientNodes.GenerateJwtAssertion(
    privateKeyPem,
    clientId,
    "https://developer.api.autodesk.com/",
    ["data:read", "data:write"],
    60
)

// Verify JWT was generated (should be long string)
if jwt != null && jwt.Length > 100:
    // JWT generated successfully
```

```
// Try token exchange
tokenBody = "grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer&assertion=" +
jwt
response = ExecuteNodes.POST(client, tokenEndpoint, tokenBody)

if not response.IsSuccessful:
    error = JsonNodes.GetContent(response)
    // Check error message for specific issue
```

Common Causes:

Wrong Client ID:

```
// Verify clientId matches your service account application
// For Autodesk: Check APS portal for correct client ID
// For Google: Check service account JSON file
```

Wrong Audience URL:

```
// Must match exactly what API expects
// Autodesk APS: "https://developer.api.autodesk.com/"
// Google: "https://oauth2.googleapis.com/token"
// Include trailing slash if required
```

Wrong Scopes:

```
// Verify scopes are valid for the API
// Autodesk APS examples:
// - "data:read"
// - "data:write"
// - "bucket:create"

// Check API documentation for valid scope strings
```

Expired JWT:

```
// JWTs expire quickly (max 60 minutes)
// Regenerate if more than expirationMinutes have passed
current_time = DateTime.Now
if current_time > jwt_generation_time + expiration_minutes:
```

```
    // Generate fresh JWT
    new_jwt = ClientNodes.GenerateJwtAssertion(...)
```

Service Account Permission Issues

Error: 403 Forbidden with valid token

Solutions:

- **Check Account Permissions**: Service account needs proper roles/permissions
- **For Autodesk APS**: Verify service account is added to the project/hub
- **For Google**: Check IAM roles assigned to service account
- **Scope Mismatch**: Ensure JWT scopes match what you're trying to access

JWT Best Practices for Troubleshooting

```
// 1. Test JWT generation separately
jwt = ClientNodes.GenerateJwtAssertion(privateKeyPem, clientId, audience, scopes,
60)
if jwt == null || jwt.Length < 100:
    // JWT generation failed - check key format

// 2. Test token exchange separately
tokenBody = "grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer&assertion=" +
jwt
response = ExecuteNodes.POST(client, tokenEndpoint, tokenBody)
if not response.IsSuccessful:
    error_content = JsonNodes.GetContent(response)
    // Examine error details

// 3. Verify token works
tokenDict = JsonNodes.ToDictionary(response)
accessToken = Dictionary.ValueAtKey(tokenDict, "access_token")
if accessToken != null:
    // Token retrieved successfully
    ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer " + accessToken)
    test_response = ExecuteNodes.GET(client, test_endpoint)
    // Should return 200 if everything is working
```

JSON Processing Issues

"JSON parsing failed" Errors

Error: Cannot convert response to Dictionary/List

Diagnostic Steps:

```
response = ExecuteNodes.GET(client, url)

// Check if response is successful
if not response.IsSuccessful:
    error = response.ErrorMessage
    // API returned error, not JSON

// Check if content is valid JSON
if JsonNodes.IsValid(response):
    data = JsonNodes.ToDictionary(response)
else:
    content = JsonNodes.GetContent(response)
    // Examine raw content to see what was returned
```

Common Causes:

HTML Error Pages:

```
// API returned HTML error page instead of JSON
raw_content = JsonNodes.GetContent(response)
// Will show HTML like "<html><body>404 Not Found</body></html>"
```

Empty Response:

```
// API returned empty body
content = JsonNodes.GetContent(response)
if content == "" or content == null:
    // Handle empty response case
```

Invalid JSON Syntax:

```
// Malformed JSON from API
raw_content = JsonNodes.GetContent(response)
// Check for syntax errors like missing quotes, brackets
```

Wrong JSON Method Used

Problem: Using ToDictionary() on JSON array or vice versa

Solutions:

```
```

```
// For JSON objects
{
  "name": "John",
  "age": 30
}
data = JsonNodes.ToDictionary(response)

// For JSON arrays
[
  {"name": "John"},
  {"name": "Jane"}
]
data = JsonNodes.ToList(response)

// Unknown format - check first
content = JsonNodes.GetContent(response)
if content.StartsWith("["):
    // Array
    data = JsonNodes.ToList(response)
elif content.StartsWith("{"):
    // Object
    data = JsonNodes.ToDictionary(response)
```

Nested JSON Access Issues

Problem: Cannot access nested properties

Solution:

```
response_dict = JsonNodes.ToDictionary(response)

// Access nested properties step by step
user_data = response_dict["user"]
address_data = user_data["address"]
street = address_data["street"]

// Or check for existence first
if response_dict.ContainsKey("user"):
    user_data = response_dict["user"]
    if user_data.ContainsKey("address"):
        address_data = user_data["address"]
```

HTTP Status Code Issues

404 Not Found

Error: HTTP 404 status code

Solutions:

1. **Check URL**: Verify endpoint path is correct
2. **Check Base URL**: If using base URL, ensure it's correct
3. **Check Resource ID**: For URLs like `/users/123`, verify ID exists
4. **Check API Version**: Endpoint might have moved in new API version

Debugging:

```
// Test base endpoint first
response = ExecuteNodes.GET(client, "https://api.example.com/")
if response.IsSuccessful:
    // Base URL works, check specific endpoint
    response = ExecuteNodes.GET(client, "https://api.example.com/users")
```

500 Internal Server Error

Error: HTTP 500-599 status codes

Solutions:

1. **Retry**: Server errors are often temporary
2. **Check API Status**: Visit API provider's status page
3. **Wait and Retry**: Implement exponential backoff
4. **Contact Support**: If persistent, contact API provider

Retry Logic:

```
max_retries = 3
for attempt in range(max_retries):
    response = ExecuteNodes.GET(client, url)
    if response.IsSuccessful:
        break
    elif response.StatusCode >= 500:
        wait_time = 2^attempt  // 1, 2, 4 seconds
        // Implement wait logic
    else:
        break  // Don't retry client errors (4xx)
```

429 Rate Limited

Error: HTTP 429 status code

Solutions:

```
response = ExecuteNodes.GET(client, url)
if response.StatusCode == 429:
    // Check for Retry-After header
    retry_after = response.Headers.get("Retry-After", 60)
    // Wait specified time before retrying
```

Prevention:

- Add delays between requests
- Use batch endpoints when available
- Cache responses when appropriate
- Monitor API usage

POST/PUT Request Issues

"Content-Type" Errors

Problem: API rejects POST requests

Solutions:

```
// DynaFetch automatically sets Content-Type for JSON
// But some APIs may need specific format

// Check what Content-Type is being sent
response = ExecuteNodes.POST(client, "https://httpbin.org/post", json_data)
echo_data = JsonNodes.ToDictionary(response)
headers_sent = echo_data["headers"]
// Check headers_sent["Content-Type"]
```

For APIs requiring specific Content-Type:

```
// Use RequestNodes for custom Content-Type
request = RequestNodes.ByUrl(url)
request = RequestNodes.AddHeader(request, "Content-Type",
"application/vnd.api+json")
request = RequestNodes.AddJsonBody(request, json_data)
response = ExecuteNodes.Execute(client, request)
```

Data Format Issues

Problem: API rejects submitted data

Solutions:

```
// Ensure data is proper JSON format
data_dict = Dictionary.ByKeysValues(
    ["name", "email"],
    ["John Doe", "john@example.com"]
)

// Convert to JSON properly
json_string = JsonNodes.DictionaryToJson(data_dict)

// Validate JSON before sending
if JsonNodes.IsValid(json_string):
    response = ExecuteNodes.POST(client, url, json_string)
else:
    // Fix data format issue
```

Check API Documentation:

- Required fields
- Data types (string vs number)
- Date formats
- Nested object structure

415 Unsupported Media Type for PATCH

Error: HTTP 415 when sending PATCH requests

Cause: API requires specialized Content-Type like `application/merge-patch+json` but DynaFetch is sending `application/json`

Diagnostic:

```
// Test what Content-Type is being sent
response = ExecuteNodes.PATCH(client, "https://httpbin.org/patch", jsonData)
echo_data = JsonNodes.ToDictionary(response)
content_type = echo_data["headers"]["Content-Type"]
// Check if it matches what API expects
```

Solution - Use RequestNodes.AddTextContent:

```

```

```
// WRONG - AddJsonBody overrides custom Content-Type
request = RequestNodes.ByUrl(url)
request = RequestNodes.AddHeader(request, "Content-Type", "application/merge-
patch+json")
request = RequestNodes.AddJsonBody(request, jsonData)  // Resets to
application/json!
response = ExecuteNodes.PATCH(client, "", request)

// CORRECT - AddTextContent preserves custom Content-Type
client = ClientNodes.Create()
ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer " + token)

request = RequestNodes.ByUrl(url)
request = RequestNodes.AddTextContent(request, jsonData, "application/merge-
patch+json")
response = ExecuteNodes.PATCH(client, "", request)
```

When You Need Custom Content-Type:

- RFC 7396 JSON Merge Patch: `application/merge-patch+json`
- RFC 6902 JSON Patch: `application/json-patch+json`
- API-specific formats: Check API documentation

400 Bad Request with PATCH Operations

Error: HTTP 400 when sending PATCH requests

Common Causes & Solutions:

ID in Request Body:

```
// WRONG - ID is in both URL and body
url = "https://api.example.com/rooms/328"
body = '{"id": 328, "name": "New Name", "status": "active"}'
response = ExecuteNodes.PATCH(client, url, body)

// CORRECT - Remove ID from body (it's in the URL)
url = "https://api.example.com/rooms/328"
data = Dictionary.ByKeysValues(["name", "status"], ["New Name", "active"])
body = JsonNodes.DictionaryToJson(data)
response = ExecuteNodes.PATCH(client, url, body)

// Or remove ID from existing dictionary
originalData = Dictionary.RemoveKeys(fullData, ["id"])
body = JsonNodes.DictionaryToJson(originalData)
```

Wrong Field Names:

```
// Verify field names match API schema exactly
// Case-sensitive: "Name" vs "name"
// Spelling: "email" vs "Email" vs "e_mail"

// Get a record first to see correct field names
getResponse = ExecuteNodes.GET(client, url)
existingData = JsonNodes.ToDictionary(getResponse)
// Use same field names from GET response
```

Wrong Data Types:

```
// API expects strings but you're sending numbers
// WRONG
data = Dictionary.ByKeysValues(["note"], [5301.99])  // Number

// CORRECT - Convert to string
noteValue = String.FromObject(5301.99)
data = Dictionary.ByKeysValues(["note"], [noteValue])  // String

// Or if API expects number but you have string
// Use: Number.FromString() to convert
```

Missing Required Fields:

```
// Some APIs require certain fields even in PATCH
// Check API documentation for required fields
// Test with minimal PATCH first, add fields incrementally
```

Example: Complete PATCH Workflow with Error Handling:

```
// 1. Get existing resource
getResponse = ExecuteNodes.GET(client, baseUrl + "/" + resourceId)
if not getResponse.IsSuccessful:
    // Handle GET error

// 2. Prepare update data
originalData = JsonNodes.ToDictionary(getResponse)
updateData = Dictionary.RemoveKeys(originalData, ["id"])  // Remove ID
updateData = Dictionary.SetValueAtKey(updateData, "name", "New Name")  // Update
fields
```

```
// 3. Convert to JSON
jsonString = JsonNodes.DictionaryToJson(updateData)

// 4. Execute PATCH with custom content-type if needed
request = RequestNodes.ByUrl(baseUrl + "/" + resourceId)
request = RequestNodes.AddTextContent(request, jsonString, "application/merge-
patch+json")
response = ExecuteNodes.PATCH(client, "", request)

// 5. Verify success
if response.IsSuccessful:
    updatedData = JsonNodes.ToDictionary(response)
    // Success
else:
    error = JsonNodes.GetContent(response)
    // Handle specific error codes
    if response.StatusCode == 400:
        // Check for ID in body, field names, data types
    elif response.StatusCode == 415:
        // Check Content-Type header
```

Large Data Submission

Problem: Timeouts when submitting large data

Solutions:

```
// Increase timeout for large data
client = ClientNodes.Create()
ClientNodes.SetTimeout(client, 600)  // 10 minutes

// Or break into smaller chunks if API supports it
```

File Upload Issues

File Upload Fails with 400 Bad Request

Error: HTTP 400 when uploading files

Common Causes & Solutions:

Missing Field Name:

```
// API expects specific field name (e.g., "file", "image", "document")
// Check API documentation for correct field name
```

```
// WRONG - empty or wrong field name
request = RequestNodes.AddFile(request, "", filePath)

// RIGHT - correct field name
request = RequestNodes.AddFile(request, "file", filePath)
```

Wrong Content-Type:

```
// Some APIs require specific content types
// Check API documentation

// Let DynaFetch auto-detect
request = RequestNodes.AddFile(request, "file", filePath)

// Or specify explicitly
request = RequestNodes.AddFile(request, "image", filePath, "image/png")
```

Missing Authentication:

```
// File uploads usually require authentication
client = ClientNodes.Create()
ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer " + token)

request = RequestNodes.ByUrl(uploadUrl)
request = RequestNodes.AddFile(request, "file", filePath)
response = ExecuteNodes.POST(client, "", request)
```

413 Payload Too Large

Error: HTTP 413 status code

Solutions:

1. **Check File Size**: Verify file is within API limits
2. **Check API Limits**: Most APIs have max file size (check docs)
3. **Compress Files**: Reduce file size if possible
4. **Use Chunked Upload**: If API supports it

```
// Check file size before upload
fileInfo = File.GetFileInfo(filePath)
fileSizeMB = fileInfo.Length / 1048576
```

```
if fileSizeMB > 10:  // API limit is 10MB
    // Handle oversized file
else:
    // Proceed with upload
```

415 Unsupported Media Type

Error: HTTP 415 status code

Solutions:

Wrong Content-Type:

```
// Specify correct MIME type for file
request = RequestNodes.AddFile(request, "file", filePath, "image/jpeg")

// Or check what API expects
// Common types: "image/jpeg", "image/png", "application/pdf", "text/csv"
```

API Expects Different Format:

```
// Some APIs want base64-encoded content instead of multipart
// Check API documentation for upload method
```

File Not Found Errors

Error: File path doesn't exist

Solutions:

```
// Always verify file exists before upload
filePath = "C:\\Documents\\file.pdf"

if File.Exists(filePath):
    request = RequestNodes.AddFile(request, "file", filePath)
    response = ExecuteNodes.POST(client, uploadUrl, request)
else:
    // Handle missing file
    error_message = "File not found: " + filePath
```

Common Path Issues:

- Use full absolute paths: `C:\Documents\file.pdf`

- Escape backslashes in strings: `"C:\\Documents\\file.pdf"`
- Verify file hasn't been moved or deleted
- Check file permissions

Large File Upload Timeouts

Error: Request times out during upload

Solutions:

```
// Increase timeout for large files
client = ClientNodes.Create()
ClientNodes.SetTimeout(client, 600)  // 10 minutes

// Rule of thumb: 1 minute per 10MB at slow speeds
// 100MB file = 10 minute timeout minimum
```

Calculate appropriate timeout:

```
fileInfo = File.GetFileInfo(filePath)
fileSizeMB = fileInfo.Length / 1048576
timeoutSeconds = Math.Max(300, fileSizeMB * 6)  // 6 seconds per MB, min 5 minutes

client = ClientNodes.Create()
ClientNodes.SetTimeout(client, timeoutSeconds)
```

File Upload with Metadata Issues

Problem: Metadata not being sent correctly

Solutions:

```
// Add metadata fields after creating file upload
formData = RequestNodes.CreateFileUpload(filePath, "file")
formData = RequestNodes.AddFormField(formData, "description", "Project photo")
formData = RequestNodes.AddFormField(formData, "category", "Construction")

response = ExecuteNodes.POST(client, uploadUrl, formData)

// Verify metadata was received (test with httpbin.org)
response = ExecuteNodes.POST(client, "https://httpbin.org/post", formData)
echo_data = JsonNodes.ToDictionary(response)
// Check echo_data["form"] for metadata fields
```

File Upload Progress Issues

Problem: Cannot track upload progress

Current Limitation: DynaFetch doesn't support progress callbacks for uploads

Workarounds:

- Display "Uploading..." message before calling POST
- Use smaller files to reduce wait time
- Implement timeout warnings for large files
- Consider API's async upload endpoints if available

Performance Issues

Slow Response Times

Problem: API calls taking too long

Diagnostic Steps:

1. **Test API Directly**: Use browser or Postman to test API speed
2. **Check Network**: Test internet connection speed
3. **Monitor API Status**: Check if API is experiencing issues

Solutions:

```
// Set appropriate timeout
ClientNodes.SetTimeout(client, 120)  // 2 minutes

// Use specific endpoints instead of broad queries
// Instead of: GET /users (all users)
// Use: GET /users?limit=10 (limited results)
```

Memory Issues with Large Responses

Problem: Dynamo becomes slow with large JSON responses

Solutions:

```
// Check response size first
content = JsonNodes.GetContent(response)
content_size = content.Length
if content_size > 1000000:  // > 1MB
    // Consider pagination or filtering
```

Pagination Strategy:

```
// Request data in pages
page = 1
all_data = []
while True:
    response = ExecuteNodes.GET(client, f"/api/data?page={page}&limit=100")
    page_data = JsonNodes.ToList(response)
    if page_data.Count == 0:
        break
    all_data.extend(page_data)
    page += 1
```

Advanced Troubleshooting

Debug HTTP Traffic

Using httpbin.org for testing:

```
// Test what headers are being sent
client = ClientNodes.Create()
ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer test-token")
response = ExecuteNodes.GET(client, "https://httpbin.org/headers")
headers_data = JsonNodes.ToDictionary(response)
// Examine headers_data["headers"] to see what was sent
```

Test POST data:

```
test_data = '{"name":"John","email":"john@example.com"}'
response = ExecuteNodes.POST(client, "https://httpbin.org/post", test_data)
echo_data = JsonNodes.ToDictionary(response)
// Check echo_data["json"] to see what was received
```

Network Diagnostics

Test connectivity:

```
// Test basic HTTP
response = ExecuteNodes.GET(client, "https://httpbin.org/get")

// Test HTTPS
response = ExecuteNodes.GET(client, "https://httpbin.org/get")
```

```
// Test with authentication
ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer test")
response = ExecuteNodes.GET(client, "https://httpbin.org/bearer")
```

SSL/TLS Issues

Problem: SSL certificate errors

Solutions:

1. **Check Certificate**: Verify API's SSL certificate is valid
2. **Update System**: Ensure Windows/certificates are up to date
3. **Test in Browser**: Verify SSL works in web browser
4. **Contact API Provider**: Report SSL issues

Proxy/Corporate Network Issues

Problem: Requests fail in corporate environment

Solutions:

1. **Check Proxy Settings**: Windows proxy settings affect DynaFetch
2. **Whitelist Domains**: Add API domains to proxy whitelist
3. **Test from Different Network**: Verify issue is network-specific
4. **Contact IT**: Corporate firewalls may need configuration

Version Compatibility Issues

Problem: DynaFetch behaves differently than expected

Check Versions:

- **Dynamo Version**: Ensure 3.0 or later
- **.NET Version**: DynaFetch requires .NET 8
- **Package Version**: Check if newer version available

Getting Help

Before Reporting Issues

Gather this information:

1. **Dynamo Version**: Help → About Dynamo
2. **DynaFetch Version**: Check in Package Manager
3. **Error Messages**: Exact error text
4. **Sample Graph**: Minimal example showing issue

5. **API Details**: Which API you're trying to access (if public)

Information to Include

For Connection Issues:

- URL being accessed
- Error message
- Network environment (corporate/home)

For Authentication Issues:

- Authentication method being used
- Error response from API
- Header format being sent

For JSON Issues:

- Raw response content (if not sensitive)
- Expected vs actual data structure
- JSON validation results

Community Resources

- **GitHub Issues**: Report bugs and feature requests
- **Dynamo Forums**: Community support and discussions
- **API Provider Support**: For API-specific issues

Quick Diagnostic Checklist

- ☐ Is Dynamo 3.0 or later?
- ☐ Is DynaFetch package installed and loaded?
- ☐ Is the API URL correct and accessible?
- ☐ Is authentication configured properly?
- ☐ Does the API return successful HTTP status?
- ☐ Is the response valid JSON?
- ☐ Are you using the correct JSON processing method?

Most DynaFetch issues can be resolved by following systematic troubleshooting steps. Start with the basics (connectivity, authentication) before moving to advanced diagnostics. The httpbin.org service is particularly useful for testing HTTP requests and responses.