DynaFetch API Documentation

> **Quick Reference**: For a fast lookup of all nodes as they appear in Dynamo, see the Node Library Reference

Complete method reference with detailed examples and parameters

Complete reference for all DynaFetch methods with parameters, return types, and examples.

Overview

DynaFetch provides 238 total nodes organized into five main categories:

DynaFetch Core Package (158 nodes)

- **ClientNodes**: HTTP client management and configuration (part of Core - 94 nodes)
- **RequestNodes**: Request building and customization (part of Nodes - 52 nodes)
- **ExecuteNodes**: HTTP method execution (part of Nodes - 52 nodes)
- **JsonNodes**: JSON processing and data conversion (part of Nodes - 52 nodes)
- **Utilities**: Helper methods and support functions (12 nodes)

System Integration (80 nodes)

- **Exception Handling**: .NET exception types for error management (13 nodes)
- **Network Operations**: .NET networking functionality (67 nodes)

Note: This documentation focuses on the DynaFetch Core Package (158 nodes) that users directly interact with. System integration nodes provide underlying infrastructure but are typically used automatically by DynaFetch.

ClientNodes - HTTP Client Management

Core Client Creation

`Create()`

Creates a new HTTP client with default settings.

Returns: `HttpClientWrapper` - The HTTP client instance

Example:

```
ClientNodes.Create() → client
```

Use Case: Start of every API workflow

`CreateWithBaseUrl(string baseUrl)`

Creates HTTP client with a base URL for all requests.

Parameters:

- `baseUrl` (string): Base URL for all requests from this client

Returns: `HttpClientWrapper` - Configured HTTP client

Example:

```
ClientNodes.CreateWithBaseUrl("https://api.github.com") → client
// Later: ExecuteNodes.GET(client, "/users/octocat") calls
https://api.github.com/users/octocat
```

Use Case: When working with single API that has consistent base URL

Client Configuration

`SetTimeout(HttpClientWrapper client, int timeoutSeconds)`

Sets request timeout for the client.

Parameters:

- `client` (HttpClientWrapper): The HTTP client to configure
- `timeoutSeconds` (int): Timeout in seconds (default: 100)

Returns: `HttpClientWrapper` - The same client with updated timeout

Example:

```
ClientNodes.SetTimeout(client, 30) → client
// All requests from this client will timeout after 30 seconds
```

Use Case: APIs with slow response times or large data transfers

---

`SetUserAgent(HttpClientWrapper client, string userAgent)`

Sets the User-Agent header for all requests.

Parameters:

- `client` (HttpClientWrapper): The HTTP client to configure
- `userAgent` (string): User agent string to identify your application

Returns: `HttpClientWrapper` - The same client with updated user agent

Example:

```
ClientNodes.SetUserAgent(client, "MyDynamoApp/1.0") → client
```

Use Case: API rate limiting based on user agent, analytics, debugging

---

`SetBaseUrl(HttpClientWrapper client, string baseUrl)`

Sets or updates the base URL for an existing client.

Parameters:

- `client` (HttpClientWrapper): The HTTP client to configure
- `baseUrl` (string): New base URL for requests

Returns: `HttpClientWrapper` - The same client with updated base URL

Example:

```
ClientNodes.SetBaseUrl(client, "https://api.v2.example.com") → client
```

Use Case: Switching between API versions or environments

Authentication & Headers

`AddDefaultHeader(HttpClientWrapper client, string name, string value)`

Adds a header that will be included in all requests from this client.

Parameters:

- `client` (HttpClientWrapper): The HTTP client to configure
- `name` (string): Header name (e.g., "Authorization", "X-API-Key")
- `value` (string): Header value (e.g., "Bearer token123", "api-key-value")

Returns: `HttpClientWrapper` - The same client with the new default header

Examples:

```
// Bearer token authentication
ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer " + token) → client

// API key authentication
ClientNodes.AddDefaultHeader(client, "X-API-Key", "your-api-key") → client

// Custom authentication
ClientNodes.AddDefaultHeader(client, "Custom-Auth", "custom-value") → client
```

Use Case: Any API requiring authentication; headers persist across all requests

---

`AddDefaultHeaders(HttpClientWrapper client, Dictionary<string, object> headers)`

Adds multiple headers at once.

Parameters:

- `client` (HttpClientWrapper): The HTTP client to configure
- `headers` (Dictionary<string, object>): Dictionary of header name-value pairs

Returns: `HttpClientWrapper` - The same client with all new headers

Example:

```
// Create dictionary with multiple headers
headers = Dictionary.ByKeysValues(
    ["Authorization", "X-API-Key", "Accept"],
    ["Bearer token123", "api-key", "application/json"]
)
ClientNodes.AddDefaultHeaders(client, headers) → client
```

Use Case: APIs requiring multiple authentication headers

`GetDefaultHeaders(HttpClientWrapper client)`

Returns all currently set default headers.

Parameters:

- `client` (HttpClientWrapper): The HTTP client to query

Returns: `Dictionary<string, object>` - Current default headers

Example:

```
ClientNodes.GetDefaultHeaders(client) → headers_dictionary
// Result: {"Authorization": "Bearer token123", "X-API-Key": "api-key"}
```

Use Case: Debugging authentication, verifying header configuration

---

`RemoveDefaultHeader(HttpClientWrapper client, string name)`

Removes a specific default header.

Parameters:

- `client` (HttpClientWrapper): The HTTP client to modify
- `name` (string): Name of header to remove

Returns: `HttpClientWrapper` - The same client with header removed

Example:

```
ClientNodes.RemoveDefaultHeader(client, "Authorization") → client
```

Use Case: Removing expired tokens, switching authentication methods

---

JWT Assertion Authentication

`GenerateJwtAssertion(string privateKeyPem, string clientId, string audience, List<string> scopes, int expirationMinutes)`

Generates a cryptographically signed JWT assertion for service account authentication (RFC 7523).

Parameters:

- `privateKeyPem` (string): RSA private key in PEM format (PKCS#1 or PKCS#8)
- `clientId` (string): OAuth 2.0 client ID / application ID
- `audience` (string): Token audience URL (e.g., "https://developer.api.autodesk.com/")
- `scopes` (List): List of OAuth 2.0 scope strings
- `expirationMinutes` (int): Token validity period in minutes (1-60, default: 60)

Returns: `string` - Signed JWT assertion ready for token exchange

Examples:

```
```

```
// Autodesk Platform Services (APS) Secure Service Account
jwt = ClientNodes.GenerateJwtAssertion(
    privateKeyPem,
    "your-client-id",
    "https://developer.api.autodesk.com/",
    ["data:read", "data:write"],
    60
)

// Exchange JWT for access token
tokenBody = "grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer&assertion=" +
jwt
response = ExecuteNodes.POST(client,
"https://developer.api.autodesk.com/authentication/v2/token", tokenBody)
accessToken = Dictionary.ValueAtKey(JsonNodes.ToDictionary(response),
"access_token")

// Use access token for authenticated requests
ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer " + accessToken)
```

Use Cases:

- **Autodesk APS Secure Service Accounts (SSA)**: Automated access to Autodesk Platform Services without user login
- **Google Service Accounts**: Server-to-server authentication for Google APIs
- **CI/CD Pipelines**: Automated API access in build/deployment workflows
- **Background Automation**: Scheduled tasks requiring API access without user interaction
- **Custom OAuth 2.0 JWT Flows**: Any RFC 7523-compliant JWT assertion authentication

Security Notes:

- Store private keys securely (never commit to version control)
- Use shortest practical expiration time (60 minutes max)
- Each JWT includes unique ID (jti) for replay attack prevention
- Supports both PKCS#1 (`-----BEGIN RSA PRIVATE KEY-----`) and PKCS#8 (`-----BEGIN PRIVATE KEY-----`) formats

ExecuteNodes - HTTP Method Execution

These are the most commonly used nodes for making API calls.

GET Requests

`GET(HttpClientWrapper client, string url)`

Performs HTTP GET request.

Parameters:

- `client` (HttpClientWrapper): Configured HTTP client
- `url` (string): Full URL or endpoint (if client has base URL)

Returns: `HttpResponse` - Response object with status, content, headers

Examples:

```
// Full URL
ExecuteNodes.GET(client, "https://api.github.com/users/octocat") → response

// With base URL client
base_client = ClientNodes.CreateWithBaseUrl("https://api.github.com")
ExecuteNodes.GET(base_client, "/users/octocat") → response
```

Use Case: Retrieving data from APIs, downloading JSON, fetching resources

---

POST Requests

`POST(HttpClientWrapper client, string url, string jsonData)`

Performs HTTP POST request with JSON data.

Parameters:

- `client` (HttpClientWrapper): Configured HTTP client
- `url` (string): Full URL or endpoint
- `jsonData` (string): JSON string to send in request body

Returns: `HttpResponse` - Response object with status, content, headers

Examples:

```
// Simple POST with JSON string
json_data = '{"name": "John", "email": "john@example.com"}'
ExecuteNodes.POST(client, "https://api.example.com/users", json_data) → response

// POST with Dictionary converted to JSON
data_dict = Dictionary.ByKeysValues(["name", "email"], ["John", "john@example.com"])
json_string = JsonNodes.DictionaryToJson(data_dict)
ExecuteNodes.POST(client, "https://api.example.com/users", json_string) → response
```

Use Case: Creating records, submitting forms, uploading data

PUT Requests

`PUT(HttpClientWrapper client, string url, string jsonData)`

Performs HTTP PUT request with JSON data.

Parameters:

- `client` (HttpClientWrapper): Configured HTTP client
- `url` (string): Full URL or endpoint
- `jsonData` (string): JSON string to send in request body

Returns: `HttpResponse` - Response object

Example:

```
update_data = '{"name": "John Updated", "email": "john.new@example.com"}'
ExecuteNodes.PUT(client, "https://api.example.com/users/123", update_data) →
response
```

Use Case: Updating existing records, replacing resources

DELETE Requests

`DELETE(HttpClientWrapper client, string url)`

Performs HTTP DELETE request.

Parameters:

- `client` (HttpClientWrapper): Configured HTTP client
- `url` (string): Full URL or endpoint

Returns: `HttpResponse` - Response object

Example:

```
ExecuteNodes.DELETE(client, "https://api.example.com/users/123") → response
```

Use Case: Deleting records, removing resources

PATCH Requests

`PATCH(HttpClientWrapper client, string url, string jsonData)`

Performs HTTP PATCH request with JSON data. Supports both standard JSON PATCH and specialized content types like RFC 7396 Merge Patch.

Parameters:

- `client` (HttpClientWrapper): Configured HTTP client
- `url` (string): Full URL or endpoint
- `jsonData` (string): JSON string with partial update data

Returns: `HttpResponse` - Response object

Standard JSON PATCH Example (default `application/json`):

```
partial_update = '{"status": "active"}'
ExecuteNodes.PATCH(client, "https://api.example.com/users/123", partial_update) →
response
```

Custom Content-Type Example (RFC 7396 Merge Patch):

Some APIs require specialized PATCH formats like `application/merge-patch+json`. Use `RequestNodes.AddTextContent` to set custom Content-Type:

```
// 1. Create authenticated client
client = ClientNodes.Create()
ClientNodes.AddDefaultHeader(client, "Authorization", "Basic " + credentials)

// 2. Add custom Content-Type header
ClientNodes.AddDefaultHeader(client, "Content-Type", "application/merge-patch+json")

// 3. Prepare JSON merge patch data
mergeData = '{"name": "Updated Name", "note": "Updated Note"}'

// 4. Add content with explicit content type
request = RequestNodes.ByUrl("https://api.example.com/rooms/123")
request = RequestNodes.AddTextContent(request, mergeData, "application/merge-
patch+json")

// 5. Execute PATCH
response = ExecuteNodes.PATCH(client, "", request)
```

Important: When using custom Content-Types, use `RequestNodes.AddTextContent` instead of `RequestNodes.AddJsonBody`. The `AddJsonBody` node automatically sets `application/json` and will override your custom Content-Type header.

Use Cases:

- Partial updates with standard JSON PATCH
- RFC 7396 JSON Merge Patch operations
- RFC 6902 JSON Patch with operations array
- API-specific PATCH formats

JsonNodes - JSON Processing & Data Conversion

Response Processing

`ToDictionary(HttpResponse response)`

Converts JSON response to Dynamo Dictionary.

Parameters:

- `response` (HttpResponse): HTTP response containing JSON

Returns: `Dictionary<string, object>` - Dynamo Dictionary with JSON data

Example:

```
response = ExecuteNodes.GET(client, "https://api.github.com/users/octocat")
JsonNodes.ToDictionary(response) → user_dict
// Access: user_dict["login"], user_dict["name"], user_dict["public_repos"]
```

Use Case: Converting API response objects to Dynamo-friendly format

---

`ToList(HttpResponse response)`

Converts JSON array response to Dynamo List.

Parameters:

- `response` (HttpResponse): HTTP response containing JSON array

Returns: `List<object>` - Dynamo List with array elements

Example:

```
response = ExecuteNodes.GET(client, "https://jsonplaceholder.typicode.com/posts")
JsonNodes.ToList(response) → posts_list
// Access: posts_list[0], posts_list[1], etc.
```

Use Case: Converting API response arrays to Dynamo-friendly lists

---

`TryToDictionary(HttpResponse response)`

Safely converts JSON to Dictionary, returns null if conversion fails.

Parameters:

- `response` (HttpResponse): HTTP response that might contain JSON

Returns: `Dictionary<string, object>` or `null` - Dictionary if successful, null if failed

Example:

```
result = JsonNodes.TryToDictionary(response)
// Check if result is null before using
```

Use Case: Handling responses that might not be valid JSON

`TryToList(HttpResponse response)`

Safely converts JSON array to List, returns null if conversion fails.

Parameters:

- `response` (HttpResponse): HTTP response that might contain JSON array

Returns: `List<object>` or `null` - List if successful, null if failed

Use Case: Handling responses that might not be valid JSON arrays

---

Response Information

`GetContent(HttpResponse response)`

Gets the raw response content as string.

Parameters:

- `response` (HttpResponse): HTTP response

Returns: `string` - Raw response content

Example:

```
raw_content = JsonNodes.GetContent(response)
// Raw content: '{"name":"John","email":"john@example.com"}'
```

Use Case: Debugging responses, handling non-JSON content, error analysis

---

`Format(HttpResponse response)`

Pretty-prints JSON response for readability.

Parameters:

- `response` (HttpResponse): HTTP response containing JSON

Returns: `string` - Formatted JSON string

Example:

```
formatted = JsonNodes.Format(response)
// Result:
// {
//   "name": "John",
//   "email": "john@example.com"
// }
```

Use Case: Debugging, displaying JSON in readable format

`IsValid(HttpResponse response)`

Checks if response contains valid JSON.

Parameters:

- `response` (HttpResponse): HTTP response to validate

Returns: `bool` - True if valid JSON, false otherwise

Example:

```
is_json = JsonNodes.IsValid(response)
// Use in conditional logic before processing JSON
```

Use Case: Validation before JSON processing, error handling

Data Conversion (Static Methods)

`JsonToDictionary(string json)`

Converts JSON string to Dictionary.

Parameters:

- `json` (string): JSON string

Returns: `Dictionary<string, object>` - Dynamo Dictionary

Example:

```
json_string = '{"name":"John","age":30}'
JsonNodes.JsonToDictionary(json_string) → dictionary
```

Use Case: Converting JSON strings from any source to Dynamo format

---

`DictionaryToJson(Dictionary<string, object> dictionary)`

Converts Dynamo Dictionary to JSON string.

Parameters:

- `dictionary` (Dictionary<string, object>): Dynamo Dictionary

Returns: `string` - JSON string

Example:

```
data = Dictionary.ByKeysValues(["name", "age"], ["John", 30])
JsonNodes.DictionaryToJson(data) → '{"name":"John","age":30}'
```

Use Case: Preparing Dynamo data for API submission

`Serialize(object data)`

Converts any object to JSON string.

Parameters:

- `data` (object): Any serializable object

Returns: `string` - JSON representation

Use Case: Converting complex objects to JSON

`Deserialize(string json)`

Converts JSON string to object.

Parameters:

- `json` (string): JSON string

Returns: `object` - Deserialized object

Use Case: Converting JSON to .NET objects

`TryDeserialize(string json)`

Safely deserializes JSON, returns null if it fails.

Parameters:

- `json` (string): JSON string

Returns: `object` or `null` - Deserialized object or null

Use Case: Safe JSON parsing without exceptions

RequestNodes - Advanced Request Building

RequestNodes provide fine-grained control over request construction. Most users will prefer the simpler ExecuteNodes, but RequestNodes are available for advanced scenarios.

Request Creation

`ByUrl(string url)`

Creates a new HTTP request for the specified URL.

Parameters:

- `url` (string): Target URL

Returns: `HttpRequest` - Request builder object

`ByEndpoint(string endpoint)`

Creates request for an endpoint (requires base URL in client).

Parameters:

- `endpoint` (string): API endpoint path

Returns: `HttpRequest` - Request builder object

Request Customization

`AddHeader(HttpRequest request, string name, string value)`

Adds a header to this specific request.

Parameters:

- `request` (HttpRequest): Request to modify
- `name` (string): Header name
- `value` (string): Header value

Returns: `HttpRequest` - Modified request

`AddBearerToken(HttpRequest request, string token)`

Adds Bearer token authorization to this request.

Parameters:

- `request` (HttpRequest): Request to modify
- `token` (string): Bearer token

Returns: `HttpRequest` - Request with authorization

`AddJsonBody(HttpRequest request, string json)`

Sets JSON body for the request.

Parameters:

- `request` (HttpRequest): Request to modify
- `json` (string): JSON string body

Returns: `HttpRequest` - Request with JSON body

File Upload Methods

`AddFile(HttpRequest request, string fieldName, string filePath, string? contentType = null)`

Adds a file to the request for multipart form-data upload.

Parameters:

- `request` (HttpRequest): Request to modify
- `fieldName` (string): Form field name (e.g., "file", "image", or API-specific field name)
- `filePath` (string): Full path to file on disk
- `contentType` (string, optional): MIME type (e.g., "image/png", "application/pdf"). Auto-detected from extension if not provided.

Returns: `HttpRequest` - Request with file attached, ready for method chaining

Example:

```
// Single file upload
request = RequestNodes.ByUrl("https://api.example.com/upload")
request = RequestNodes.AddFile(request, "image", "C:\\Photos\\photo.jpg",
"image/jpeg")
response = ExecuteNodes.POST(client, "", request)

// File upload with authentication
request = RequestNodes.ByUrl(uploadUrl)
request = RequestNodes.AddHeader(request, "Authorization", "Bearer " + token)
request = RequestNodes.AddFile(request, "file", filePath, "image/png")
response = ExecuteNodes.POST(client, "", request)
```

Use Cases:

- API image/document uploads
- BIMtrack project image uploads
- Autodesk APS file storage
- Any multipart/form-data file upload scenario

Supported MIME Types: Automatically detected for common extensions (jpg, png, gif, pdf, doc, docx, xls, xlsx, txt, csv, zip, etc.)

`CreateFileUpload(string filePath, string? fieldName = null, string? fileName = null, string? contentType = null)`

Creates multipart form-data content directly for file uploads (alternative to AddFile).

Parameters:

- `filePath` (string): Full path to file on disk

- `fieldName` (string, optional): Form field name. Defaults to empty string if not provided.
- `fileName` (string, optional): Custom filename to use. Defaults to actual filename if not provided.
- `contentType` (string, optional): MIME type. Auto-detected from file extension if not provided.

Returns: `MultipartFormDataContent` (as object) - Ready for POST/PUT/PATCH

Example:

```
// Simple file upload
formData = RequestNodes.CreateFileUpload("C:\\Documents\\file.pdf", "document")
response = ExecuteNodes.POST(client, "https://api.example.com/upload", formData)

// With custom filename and content type
formData = RequestNodes.CreateFileUpload(
    "C:\\temp\\data.bin",
    "datafile",
    "custom_name.bin",
    "application/octet-stream"
)
response = ExecuteNodes.POST(client, uploadUrl, formData)
```

Use Case: When you need direct multipart content without request chaining

---

`AddFormField(MultipartFormDataContent formData, string fieldName, string value)`

Adds text fields to existing multipart form-data (use with CreateFileUpload).

Parameters:

- `formData` (MultipartFormDataContent): Existing form data from CreateFileUpload
- `fieldName` (string): Name of the form field
- `value` (string): Text value for the field

Returns: `MultipartFormDataContent` - Updated form data with new field

Example:

```
// File upload with metadata
formData = RequestNodes.CreateFileUpload("C:\\image.jpg", "file")
formData = RequestNodes.AddFormField(formData, "description", "Project photo")
formData = RequestNodes.AddFormField(formData, "category", "Construction")
response = ExecuteNodes.POST(client, uploadUrl, formData)
```

Use Case: Adding metadata, descriptions, or parameters alongside file uploads

---

HttpResponse Properties

Every ExecuteNodes method returns an `HttpResponse` object with these properties:

- `IsSuccessful` (bool): True if HTTP status 200-299
- `StatusCode` (int): HTTP status code (200, 404, 500, etc.)
- `Content` (string): Raw response content
- `Headers` (Dictionary): Response headers
- `ErrorMessage` (string): Error description if request failed

Example Usage:

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

if response.IsSuccessful:
    data = JsonNodes.ToDictionary(response)
    // Process successful response
else:
    error = response.ErrorMessage
    // Handle error
```

Common Patterns

Pattern 1: Simple API Call

```
client = ClientNodes.Create()
response = ExecuteNodes.GET(client, "https://api.example.com/data")
data = JsonNodes.ToDictionary(response)
```

Pattern 2: Authenticated API Call

```
client = ClientNodes.Create()
ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer " + token)
response = ExecuteNodes.GET(client, "https://api.example.com/protected")
data = JsonNodes.ToDictionary(response)
```

Pattern 3: Data Submission

```
client = ClientNodes.Create()
ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer " + token)
```

```
json_data = JsonNodes.DictionaryToJson(your_data)
response = ExecuteNodes.POST(client, "https://api.example.com/records", json_data)
result = JsonNodes.ToDictionary(response)
```

Pattern 4: Error Handling

```
response = ExecuteNodes.GET(client, url)
if response.IsSuccessful:
    if JsonNodes.IsValid(response):
        data = JsonNodes.ToDictionary(response)
        // Success
    else:
        // Not JSON response
else:
    error = response.ErrorMessage
    // Handle HTTP error
```

Pattern 5: Custom Content-Type PATCH (RFC 7396 Merge Patch)

```
// Setup authenticated client
client = ClientNodes.Create()
ClientNodes.AddDefaultHeader(client, "Authorization", "Basic " + credentials)

// Prepare data - remove ID fields, only include fields to update
updateData = Dictionary.ByKeysValues(["name", "note"], ["New Name", "New Note"])
jsonString = JsonNodes.DictionaryToJson(updateData)

// Build request with custom content type
request = RequestNodes.ByUrl("https://api.example.com/resources/123")
request = RequestNodes.AddTextContent(request, jsonString, "application/merge-
patch+json")

// Execute PATCH
response = ExecuteNodes.PATCH(client, "", request)
result = JsonNodes.ToDictionary(response)
```

Pattern 6: JWT Service Account Authentication

```
// 1. Generate JWT assertion
jwt = ClientNodes.GenerateJwtAssertion(
    privateKeyPem,
    clientId,
    "https://developer.api.autodesk.com/",
```

```
    ["data:read", "data:write"],
    60
)

// 2. Exchange JWT for access token
tokenBody = "grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer&assertion=" +
jwt
client = ClientNodes.Create()
tokenResponse = ExecuteNodes.POST(client,
"https://developer.api.autodesk.com/authentication/v2/token", tokenBody)

// 3. Extract access token
tokenDict = JsonNodes.ToDictionary(tokenResponse)
accessToken = Dictionary.ValueAtKey(tokenDict, "access_token")

// 4. Use token for authenticated requests
ClientNodes.AddDefaultHeader(client, "Authorization", "Bearer " + accessToken)
response = ExecuteNodes.GET(client,
"https://developer.api.autodesk.com/data/v1/projects")
```

Performance Tips

1. **Reuse clients**: Create once, use for multiple requests
2. **Use base URLs**: More efficient than full URLs each time
3. **Set appropriate timeouts**: Default 100s may be too long for some uses
4. **Check IsSuccessful**: Before processing JSON responses
5. **Use Try methods**: For uncertain JSON responses
6. **File uploads**: Use AddFile for method chaining, CreateFileUpload for direct multipart content
7. **JWT tokens**: Cache generated tokens - don't regenerate for every request (60-minute validity)
8. **Custom content-types**: Use RequestNodes.AddTextContent instead of AddJsonBody when setting non-standard Content-Type headers

Error Handling

Common error scenarios and how to handle them:

- **Network errors**: Check `response.IsSuccessful` and `response.ErrorMessage`
- **JSON parsing errors**: Use `JsonNodes.IsValid()` before conversion
- **Authentication errors**: Check status code 401/403, verify headers
- **Rate limiting**: Check status code 429, implement retry logic
- **Timeouts**: Increase timeout or check network connectivity

Troubleshooting

415 Unsupported Media Type Error

Symptom: API returns "415 Unsupported Media Type" status code

Cause: The API expects a specific Content-Type (like `application/merge-patch+json`) but received `application/json`

Solution: Use `RequestNodes.AddTextContent` instead of `RequestNodes.AddJsonBody` when setting custom Content-Type headers:

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

// CORRECT - AddTextContent preserves custom Content-Type
client = ClientNodes.Create()
ClientNodes.AddDefaultHeader(client, "Authorization", "Basic " + credentials)
request = RequestNodes.ByUrl(url)
request = RequestNodes.AddTextContent(request, jsonString, "application/merge-
patch+json")
response = ExecuteNodes.PATCH(client, "", request)
```

When to use custom Content-Types:

- 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

Symptom: API returns "400 Bad Request" for PATCH requests

Common Causes:

1. **ID in both URL and body**: Remove `id` field from request body - it's already in the URL endpoint
2. **Wrong field names**: Verify field names match API schema exactly
3. **Wrong data types**: Check if API expects strings vs numbers (use `String.FromObject` to convert)
4. **Missing required fields**: Some APIs require certain fields even in PATCH operations

Example Fix:

```
// Remove ID from body before PATCH
data = Dictionary.RemoveKeys(originalData, ["id"])
jsonString = JsonNodes.DictionaryToJson(data)
response = ExecuteNodes.PATCH(client, baseUrl + "/" + id, jsonString)
```

For usage examples and workflows, see the main README.md
For migration from DynaWeb, see Migration-Guide.md