DynaFetch Node Library Reference

Quick reference for primary workflow nodes - the essential nodes for building REST API workflows in Dynamo

> **Complete Reference**: For ALL 220+ nodes including Core infrastructure (100+ individual nodes), System utilities, and HTTP status codes, see the Advanced Node Library Reference

DynaFetch nodes are organized under the main **DynaFetch** category with the following structure:

```
DynaFetch/
├── Core/                    # Core HTTP infrastructure classes
├── Nodes/                   # Primary workflow nodes
│   ├── ClientNodes/         # HTTP client management
│   ├── ExecuteNodes/        # HTTP method execution
│   ├── JsonNodes/           # JSON processing utilities
│   └── RequestNodes/        # Request building and configuration
└── Utilities/
    └── JsonHelper/          # Static JSON utility methods
```

Quick Navigation

- Primary Workflow Nodes - Main nodes for building API workflows
  - ClientNodes - HTTP client creation and configuration
  - RequestNodes - Request construction and setup
  - ExecuteNodes - API calls and response handling
  - JsonNodes - Data transformation and validation
- Core Infrastructure - Advanced classes for complex scenarios
- Utilities - Static helper methods
- Common Patterns - Typical node combinations

Primary Workflow Nodes

These are the main nodes you'll use for building REST API workflows in Dynamo

ClientNodes

Create and configure HTTP clients for API connections

ClientNodes.Create

Purpose: Creates a new HTTP client for making API requests
Inputs: None
Outputs: HttpClientWrapper

Description: Starting point for all DynaFetch workflows. Creates a fresh HTTP client with default settings (30-second timeout, standard headers).

Usage Pattern:

```
ClientNodes.Create() → RequestNodes.ByUrl() → ExecuteNodes.GET()
```

ClientNodes.CreateWithBaseUrl

Purpose: Creates HTTP client with a predefined base URL
Inputs: baseUrl (string)
Outputs: HttpClientWrapper
Description: Convenient for APIs where all requests use the same domain. Set once, use relative paths for individual requests.

Example: `CreateWithBaseUrl("https://api.example.com")` then use endpoint "/users/123"

ClientNodes.SetBaseUrl

Purpose: Adds or updates the base URL for an existing client
Inputs: client (HttpClientWrapper), baseUrl (string)
Outputs: HttpClientWrapper
Description: Modify base URL after client creation. Useful for switching between development and production environments.

ClientNodes.SetTimeout

Purpose: Configures request timeout duration
Inputs: client (HttpClientWrapper), timeoutSeconds (int)
Outputs: HttpClientWrapper
Description: Prevents hanging requests. Default is 30 seconds.

Recommended Values:

- 10-15 seconds for responsive UIs
- 30 seconds for typical APIs
- 60+ seconds for slow operations or large data transfers

ClientNodes.SetUserAgent

Purpose: Sets the User-Agent header for all requests
Inputs: client (HttpClientWrapper), userAgent (string)
Outputs: HttpClientWrapper

Description: Identifies your application to APIs. Some APIs require specific user agents or block requests without proper identification.

Example: `"MyDynamoApp/1.0"`, `"DynaFetch/1.0"`, or `"CompanyName-AutomationTool/2.1"`

ClientNodes.AddDefaultHeader

Purpose: Adds a header to all requests from this client
Inputs: client (HttpClientWrapper), name (string), value (string)
Outputs: HttpClientWrapper
Description: Perfect for authentication tokens that don't change. Headers persist across all requests from this client.

Common Usage:

- `AddDefaultHeader(client, "Authorization", "Bearer your-token-here")`
- `AddDefaultHeader(client, "X-API-Key", "your-api-key")`

ClientNodes.AddDefaultHeaders

Purpose: Adds multiple headers at once
Inputs: client (HttpClientWrapper), headers (Dictionary<string, string>)
Outputs: HttpClientWrapper
Description: Bulk header addition for complex authentication or API requirements.

ClientNodes.GetDefaultHeaders

Purpose: Retrieves current default headers
Inputs: client (HttpClientWrapper)
Outputs: Dictionary<string, string>
Description: View currently configured headers for debugging, validation, or conditional logic.

ClientNodes.RemoveDefaultHeader

Purpose: Removes a specific default header
Inputs: client (HttpClientWrapper), name (string)
Outputs: HttpClientWrapper
Description: Remove headers that are no longer needed without recreating the entire client.

RequestNodes

Construct and configure HTTP requests before execution

RequestNodes.ByUrl

Purpose: Creates a request from a complete URL
Inputs: client (HttpClientWrapper), url (string)
Outputs: HttpRequest
Description: Use for complete URLs. DynaFetch handles URL validation, formatting, and encoding automatically.

Example: `ByUrl(client, "https://api.example.com/users/123?include=profile")`

RequestNodes.ByEndpoint

Purpose: Creates a request from an endpoint path
Inputs: client (HttpClientWrapper), endpoint (string)
Outputs: HttpRequest
Description: Combines with client's base URL. Useful when base URL is already configured on the client.

Example: If base URL is "https://api.example.com", use endpoint "/users/123"

RequestNodes.AddHeader

Purpose: Adds a single header to this specific request
Inputs: request (HttpRequest), name (string), value (string)
Outputs: HttpRequest
Description: Request-specific headers that don't affect other requests from the same client. Overrides default headers with the same name.

RequestNodes.AddHeaders

Purpose: Adds multiple headers to this request
Inputs: request (HttpRequest), headers (Dictionary<string, string>)
Outputs: HttpRequest
Description: Bulk header addition for requests requiring multiple custom headers.

RequestNodes.AddBearerToken

Purpose: Adds Authorization header with Bearer token
Inputs: request (HttpRequest), token (string)
Outputs: HttpRequest
Description: Shortcut for the most common authentication pattern. Automatically formats as "Bearer {token}".

RequestNodes.AddParameter

Purpose: Adds a single query parameter
Inputs: request (HttpRequest), name (string), value (string)
Outputs: HttpRequest
Description: Builds query string automatically. Handles URL encoding and proper formatting.

Example: `AddParameter(request, "limit", "10")` results in `?limit=10`

RequestNodes.AddParameters

Purpose: Adds multiple query parameters
Inputs: request (HttpRequest), parameters (Dictionary<string, string>)
Outputs: HttpRequest
Description: Bulk parameter addition. Creates properly formatted and encoded query strings.

RequestNodes.AddJsonBody

Purpose: Adds JSON content from a Dictionary
Inputs: request (HttpRequest), data (Dictionary<string, object>)
Outputs: HttpRequest
Description: Converts Dictionary to JSON automatically. Sets correct Content-Type header to "application/json".

RequestNodes.AddJsonContent

Purpose: Adds pre-formatted JSON content
Inputs: request (HttpRequest), json (string)
Outputs: HttpRequest
Description: Use when you already have a JSON string. Validates JSON format before sending to prevent errors.

RequestNodes.AddTextContent

Purpose: Adds plain text content
Inputs: request (HttpRequest), content (string), contentType (string)
Outputs: HttpRequest
Description: For non-JSON APIs or custom content types. Specify exact Content-Type needed by the API.

RequestNodes.SetMethod

Purpose: Sets the HTTP method explicitly
Inputs: request (HttpRequest), method (string)
Outputs: HttpRequest
Description: Use for non-standard methods (PATCH, HEAD, OPTIONS) or when building requests dynamically.

RequestNodes.AsGet

Purpose: Configures request as GET method
Inputs: request (HttpRequest)
Outputs: HttpRequest
Description: Explicit GET configuration. GET is default for most request creation methods.

RequestNodes.AsPost

Purpose: Configures request as POST method
Inputs: request (HttpRequest)
Outputs: HttpRequest
Description: Required for POST requests. Usually combined with body content (JSON, text, or form data).

RequestNodes.AsPut

Purpose: Configures request as PUT method
Inputs: request (HttpRequest)
Outputs: HttpRequest
Description: For complete resource updates. Often requires body content containing the full updated resource.

RequestNodes.AsDelete

Purpose: Configures request as DELETE method
Inputs: request (HttpRequest)
Outputs: HttpRequest
Description: For deletion operations. Usually no body content needed, just specify the resource to delete.

RequestNodes.AsPatch

Purpose: Configures request as PATCH method
Inputs: request (HttpRequest)
Outputs: HttpRequest
Description: For partial resource updates. Body contains only the fields that need to be updated.

RequestNodes.AddFile

Purpose: Add a file to request for upload (multipart form-data)
Inputs: request (HttpRequest), fieldName (string), filePath (string), contentType (string)
Outputs: HttpRequest
Description: Adds file to request using DynaWeb-compatible builder pattern. Returns updated HttpRequest for method chaining. Use with ExecuteNodes.POST/PUT/PATCH for file uploads. Field name is the form field (e.g., "file" or "application/json" for BIMtrack). Content type is MIME type (e.g., "image/png", "multipart/form-data").

RequestNodes.CreateFileUpload

Purpose: Create multipart form-data content for file uploads
Inputs: filePath (string), fieldName (string, optional), fileName (string, optional), contentType (string, optional)
Outputs: MultipartFormDataContent (as object)
Outputs: object (MultipartFormDataContent)
Description: Creates multipart form-data directly. Returns content ready for ExecuteNodes.POST/PUT/PATCH. Field name defaults to empty string, file name defaults to actual filename, content type auto-detected from extension if not provided. Use when not chaining with other request configuration.

RequestNodes.AddFormField

Purpose: Add text field to existing multipart form data
Inputs: formData (MultipartFormDataContent), fieldName (string), value (string)
Outputs: MultipartFormDataContent
Description: Adds additional text fields alongside file uploads. Use after CreateFileUpload to add metadata or parameters. Returns updated form data for method chaining or passing to ExecuteNodes.

ExecuteNodes

Execute HTTP requests and receive responses

ExecuteNodes.GET

Purpose: Executes GET request
Inputs: client (HttpClientWrapper), url (string)
Outputs: HttpResponse
Description: Sends GET request and returns response. Simple pattern for retrieving data from APIs.

ExecuteNodes.POST

Purpose: Executes POST request with optional content
Inputs: client (HttpClientWrapper), url (string), content (var, optional)
Outputs: HttpResponse
Description: Sends POST request. Accepts: (1) string for JSON data, (2) HttpRequest with files from AddFile, or (3) MultipartFormDataContent for file uploads. No content parameter creates simple POST.

ExecuteNodes.PUT

Purpose: Executes PUT request with optional content
Inputs: client (HttpClientWrapper), url (string), content (var, optional)
Outputs: HttpResponse
Description: Sends PUT request for resource updates. Accepts: (1) string for JSON data, (2) HttpRequest with files from AddFile, or (3) MultipartFormDataContent for file uploads.

ExecuteNodes.DELETE

Purpose: Executes DELETE request
Inputs: client (HttpClientWrapper), url (string)
Outputs: HttpResponse
Description: Sends DELETE request to remove resources. Check response StatusCode to confirm successful deletion.

ExecuteNodes.PATCH

Purpose: Executes PATCH request with optional content
Inputs: client (HttpClientWrapper), url (string), content (var, optional)
Outputs: HttpResponse
Description: Sends PATCH request for partial updates. Accepts: (1) string for JSON data, (2) HttpRequest with files from AddFile, or (3) MultipartFormDataContent for file uploads.

JsonNodes

Transform and validate API response data for use in Dynamo

Response Processing

JsonNodes.ToDictionary

Purpose: Converts JSON response to Dynamo Dictionary
Inputs: response (HttpResponse)
Outputs: Dictionary<string, object>
Description: Primary method for accessing JSON object data. Handles nested objects and arrays. Preserves data types.

Usage: Access data like `dictionary["property"]` or `dictionary["user"]["name"]` in Dynamo

JsonNodes.ToList

Purpose: Converts JSON array response to Dynamo List
Inputs: response (HttpResponse)
Outputs: List
Description: For JSON arrays. Each list item can be Dictionary (for objects), List (for nested arrays), or primitive values.

JsonNodes.ToObject

Purpose: Converts JSON to most appropriate Dynamo type
Inputs: response (HttpResponse)
Outputs: object
Description: Smart conversion - returns Dictionary for objects, List for arrays, string/number/boolean for primitives.

Safe Processing

JsonNodes.TryToDictionary

Purpose: Safe conversion to Dictionary with fallback
Inputs: response (HttpResponse)

Outputs: Dictionary<string, object> (empty if conversion fails)
Description: Won't crash your graph. Returns empty Dictionary if JSON isn't an object or if parsing fails.

JsonNodes.TryToList

Purpose: Safe conversion to List with fallback
Inputs: response (HttpResponse)
Outputs: List (empty if conversion fails)
Description: Won't crash your graph. Returns empty List if JSON isn't an array or if parsing fails.

Response Utilities

JsonNodes.Format

Purpose: Returns pretty-printed JSON for reading
Inputs: response (HttpResponse)
Outputs: string
Description: Formats JSON with proper indentation and line breaks. Useful for debugging, logging, or displaying JSON content.

JsonNodes.IsValid

Purpose: Checks if response contains valid JSON
Inputs: response (HttpResponse)
Outputs: bool
Description: Validate JSON before processing. Use to prevent errors in downstream nodes or implement conditional logic.

JsonNodes.GetContent

Purpose: Returns raw response content as string
Inputs: response (HttpResponse)
Outputs: string
Description: Access raw response content. Use for non-JSON responses, debugging, or when you need the exact server response.

JsonNodes.Deserialize

Purpose: Converts JSON string to object
Inputs: json (string)
Outputs: object
Description: For processing JSON from sources other than HTTP responses (files, user input, etc.).

JsonNodes.TryDeserialize

Purpose: Safe JSON string conversion with fallback
Inputs: json (string)
Outputs: object (null if conversion fails)
Description: Won't crash your graph. Returns null if JSON string is invalid or malformed.

Static Utilities

JsonNodes.JsonToDictionary

Purpose: Converts JSON string directly to Dictionary
Inputs: json (string)
Outputs: Dictionary<string, object>
Description: Direct string-to-Dictionary conversion. For processing saved JSON strings or user-provided JSON.

JsonNodes.DictionaryToJson

Purpose: Converts Dictionary to JSON string
Inputs: dictionary (Dictionary<string, object>)
Outputs: string
Description: Prepare Dictionary data for API submission, storage, or transmission.

JsonNodes.Serialize

Purpose: Converts any object to JSON string
Inputs: obj (object)
Outputs: string
Description: General-purpose serialization. Works with Dictionaries, Lists, primitives, and complex objects.

JsonNodes.SerializePretty

Purpose: Converts object to formatted JSON string
Inputs: obj (object)
Outputs: string
Description: Human-readable JSON with proper indentation. For display, debugging, or storage where readability matters.

Core Infrastructure

Advanced classes for complex scenarios - typically used indirectly through the workflow nodes

These classes from the **Core** section are primarily used internally by the workflow nodes, but are available for advanced scenarios:

HttpClientWrapper

The main HTTP client class that manages connections, default headers, timeouts, and base URLs. Created by `ClientNodes.Create()`.

HttpRequest

Represents a configured HTTP request with URL, method, headers, parameters, and body content. Created by `RequestNodes.ByUrl()` or `RequestNodes.ByEndpoint()`.

HttpResponse

Contains the response from an HTTP request, including status code, headers, and content. Returned by all `ExecuteNodes` methods.

Error Handling Classes

- **DynaFetchException**: Base exception for DynaFetch operations
- **DynaFetchHttpException**: HTTP-specific errors (timeouts, connection failures)
- **DynaFetchJsonException**: JSON parsing and processing errors
- **ErrorDetails**: Structured error information for debugging
- **SafeOperations**: Utilities for error-resistant operations
- **Validation**: Input validation and sanitization

Utilities

Static helper methods for advanced JSON processing

JsonHelper

Advanced JSON processing utilities available for complex scenarios. Most common operations are available through the JsonNodes workflow nodes, but JsonHelper provides additional functionality for specialized use cases.

Common Workflow Patterns

Basic GET Request (3 nodes)

```
ClientNodes.Create() → ExecuteNodes.GET(url) → JsonNodes.ToDictionary()
```

Use for: Simple API calls where you need object data

Basic GET to List (3 nodes)

```
ClientNodes.Create() → ExecuteNodes.GET(url) → JsonNodes.ToList()
```

Use for: APIs that return arrays of data

Authenticated API Call

```
ClientNodes.Create() → ClientNodes.AddDefaultHeader("Authorization", "Bearer token")
→ ExecuteNodes.GET(url) → JsonNodes.ToDictionary()
```

Use for: APIs requiring authentication on all requests

POST with JSON Data

```
ClientNodes.Create() → RequestNodes.ByUrl(url) → RequestNodes.AddJsonBody(data) →
RequestNodes.AsPost() → ExecuteNodes.POST() → JsonNodes.ToDictionary()
```

Use for: Creating new resources or submitting form data

POST Shortcut

```
ClientNodes.Create() → ExecuteNodes.POST(url, data) → JsonNodes.ToDictionary()
```

Use for: Simple POST operations with JSON data

Multi-step Authenticated Workflow

```
1. ClientNodes.Create() → ClientNodes.AddDefaultHeader(auth)
2. ExecuteNodes.GET(url) → JsonNodes.ToDictionary() → [modify data]
3. JsonNodes.DictionaryToJson(modified_data) → RequestNodes.AddJsonContent() →
ExecuteNodes.POST()
```

Use for: Complex workflows requiring data retrieval, modification, and submission

Error-Safe Processing

```
ExecuteNodes.GET(url) → JsonNodes.IsValid() → [If True: JsonNodes.ToDictionary(), If
False: handle error]
```

Use for: Robust workflows that handle API failures gracefully

Query Parameters

```
ClientNodes.Create() → RequestNodes.ByUrl(url) → RequestNodes.AddParameter("limit",
"10") → RequestNodes.AddParameter("sort", "name") → ExecuteNodes.GET() →
JsonNodes.ToList()
```

Use for: APIs requiring query parameters for filtering, pagination, or sorting

Multiple Endpoints with Base URL

```
1. ClientNodes.CreateWithBaseUrl("https://api.example.com")
2. RequestNodes.ByEndpoint("/users") → ExecuteNodes.GET() → JsonNodes.ToList()
3. RequestNodes.ByEndpoint("/products") → ExecuteNodes.GET() → JsonNodes.ToList()
```

Use for: Working with multiple endpoints from the same API

Quick Reference by Use Case

Simple API Call: `Create() → GET(url) → ToDictionary()`

Array Data: `Create() → GET(url) → ToList()`

Authentication: Use `AddDefaultHeader()` for persistent auth, `AddBearerToken()` for single requests

POST Data: Use `AddJsonBody()` for Dictionary data, `AddJsonContent()` for JSON strings, or `POST(url, data)` shortcut

Query Parameters: Use `AddParameter()` or `AddParameters()` before execution

Error Handling: Use `IsValid()` to check responses, `Try*` methods for safe conversion

Multiple Endpoints: Set `BaseUrl` once, use `ByEndpoint()` for relative paths

Complex Headers: Use `AddDefaultHeaders()` for client-level, `AddHeaders()` for request-level

Raw Content: Use `GetContent()` for non-JSON responses or debugging

Pretty JSON: Use `Format()` for readable JSON display

Performance Notes

- **Connection Reuse**: Create client once, reuse for multiple requests to the same API

- **Default Headers**: More efficient than adding headers to each individual request
- **Base URLs**: Reduce string processing by setting base URL once and using endpoints
- **JSON Engines**: Automatic fallback between System.Text.Json and Newtonsoft.Json ensures compatibility
- **Response Size**: Large responses (>1MB) may take longer to convert to Dictionary/List
- **Timeout Settings**: Adjust based on expected API response times - too short causes failures, too long causes hanging
- **Safe Methods**: `Try*` methods have slight overhead but prevent graph crashes

---

Error Prevention Tips

- **URL Validation**: DynaFetch validates and formats URLs automatically, but verify URLs are complete
- **Header Conflicts**: Request headers override default headers with the same name
- **JSON Validation**: Use `IsValid()` before processing responses from unknown or unreliable APIs
- **Safe Methods**: Use `Try*` methods when working with unpredictable data sources
- **Content Types**: JSON methods automatically set correct Content-Type headers
- **Authentication**: Use consistent header names - "Authorization" is standard for Bearer tokens
- **Query Parameters**: DynaFetch handles URL encoding, but be aware of API-specific parameter requirements
- **Response Handling**: Check HTTP status codes in addition to JSON validity for complete error handling

---

Integration with Documentation Suite

- **Detailed Examples**: See API-Documentation.md for complete method documentation with code examples
- **Migration Guide**: See Migration-Guide.md for transitioning from DynaWeb to DynaFetch
- **Installation Help**: See README.md for installation and quick start instructions
- **Troubleshooting**: See Troubleshooting.md for problem resolution and common issues
- **Best Practices**: See Best-Practices.md for security, performance, and workflow organization guidance

---

This documentation reflects the node organization as it appears in Dynamo Sandbox 3.5.2. For the most current information, always refer to the nodes as they appear in your Dynamo installation.