Team Dynamix Web Api SDK

A fluent, opinionated C# SDK for interacting with the TeamDynamix Web API.

✨ Features

- ✅ Fluent-style request builders
- ✅ Strongly typed models and builders
- ✅ Supports both basic and complex ticket operations
- ✅ Extensible and customizable
- ✅ Designed for testability and clean layering

📦 Installation

Install via NuGet:

```
dotnet add package TdxClient
```

Or reference the project directly in your solution:

```
<ProjectReference Include="..\TdxClient\TdxClient.csproj" />
```

🔧 Setup

🚀 Quick Start

Option 1: Minimal Setup (Not Recommended for ASP.NET Core)

```
var client = new TdxClient(
    tenant: "mytenant",
    webServicesBeId: "abc123",
    webServicesKey: "super-secret-key"
);

await client.AuthenticateAsync();

var ticket = await client.Tickets("244")["123456"].GetAsync();
```

Option 2: Recommended Setup with IHttpClientFactory

In Program.cs or Startup.cs:

```
builder.Services.AddHttpClient<TdxClient>((provider, httpClient) =>
{
    // Optional: Add custom handlers, policies, etc.
})
.ConfigurePrimaryHttpMessageHandler(() => new HttpClientHandler
{
    AutomaticDecompression = DecompressionMethods.GZip
| DecompressionMethods.Deflate
});
```

Then register your config:

```
builder.Services.AddSingleton(new TdxClientOptions
{
    EnableThrottleCountdownLogging = true,
    OnThrottleWait = remaining => Console.WriteLine($"Waiting
{remaining.TotalSeconds} seconds due to rate limit..."),
    OnThrottleContinue = () => Console.WriteLine("Continuing after wait.")
});

builder.Services.AddTransient<TdxClient>(provider =>
{
    var httpClient = provider.GetRequiredService<IHttpClientFactory>
().CreateClient(nameof(TdxClient));
    var options = provider.GetRequiredService<TdxClientOptions>();

    return new TdxClient(
        httpClient,
        tenant: "mytenant",
        webServicesBeId: "abc123",
        webServicesKey: "super-secret-key",
        options
    );
});
```

Then inject it in your service:

```
public class MyService
{
    private readonly TdxClient _tdx;

    public MyService(TdxClient tdx)
```

```
    {
        _tdx = tdx;
    }

    public async Task RunAsync()
    {
        await _tdx.AuthenticateAsync();

        var newTicket = new Ticket
        {
            Title = "My New Ticket",
            Description = "Created via fluent API",
            RequestorUid = Guid.Parse("..."),
            StatusID = 1234,
            TypeID = 5678,
            AccountID = 9876,
            PriorityID = 1111
        };

        var created = await _tdx.Tickets("244").Create(newTicket);
    }
}
```

Inject Your Own HttpClient With Custom Delegating Handlers

```
// Custom logging handler example
public class LoggingHandler : DelegatingHandler
{
    protected override async Task<HttpResponseMessage> SendAsync(HttpRequestMessage
request, CancellationToken cancellationToken)
    {
        Console.WriteLine($"Request: {request.Method} {request.RequestUri}");
        var response = await base.SendAsync(request, cancellationToken);
        Console.WriteLine($"Response: {response.StatusCode}");
        return response;
    }
}

// Create a list of custom handlers
var handlers = new List<DelegatingHandler> { new LoggingHandler() };

// Create HttpClient with custom handlers using the factory
var httpClient = TdxClientFactory.CreateHttpClient(handlers, tenant);
```

```
// Inject your HttpClient into TdxClient
var client = new TdxClient(httpClient, tenant, beId, webServicesKey);

// Authenticate and use
await client.AuthenticateAsync();

var tickets = await client.Tickets("244")["12345"].GetAsync();
```

Advanced: Adding Multiple Handlers (Logging + Retry + Telemetry)

```
var handlers = new List<DelegatingHandler>
{
    new LoggingHandler(),
    new RetryHandler(),  // Your custom retry handler if you have one
    new TelemetryHandler() // Your telemetry handler for metrics
};

var httpClient = TdxClientFactory.CreateHttpClient(handlers, tenant);

var client = new TdxClient(httpClient, tenant, beId, webServicesKey);

await client.AuthenticateAsync();

// Now all requests go through the pipeline of your handlers
var tickets = await client.Tickets("244")["12345"].GetAsync();
```

⚠️ Not Recommended: Manual `HttpClient` Instantiation

You will need to handle your own retry policies.

```
// Use only for quick tests or console apps
var httpClient = new HttpClient
{
    BaseAddress = new Uri("https://your-org.teamdynamix.com/TDWebApi/api/")
};

var tdxClient = new TdxClient(httpClient, "your-webservices-key", "your-beid");
```

This bypasses DI scopes, Polly retry policies, DNS refresh handling, and other best practices.

🚀 Usage Exmaple

🔍 Search for a Person

```
var results = await tdxClient.Users.Search
    .WithEmail("jdoe@yourorg.edu")
    .ExecuteAsync();
```

📄 Get a Ticket by ID

```
var ticket = await tdxClient.Tickets("244")["123456"].GetAsync();
```

📝 Create a Ticket

```
var ticketRequest = new TicketRequest
{
    Title = "New Hire Onboarding",
    RequestorUid = "abc-123",
    Type = TicketType.OnboardingOrJobUpdate,
    Description = "Provision access and equipment",
    ResponsibleGroupId = "4000"
};

var ticket = await tdxClient.Tickets("244").Create(ticketRequest.MapToTicket());
```

📝 Create ticket cont.

```
var ticket = await tdxClient.Tickets("244").Create().WithTitle("...")...
// if you attempt to send without all required fields it will error our and
notify you
```

✅ Resolve a Ticket

```
await tdxClient.Tickets("244")["123456"]
    .SetStatus(TicketStatus.Resolved)
    .WithComment("Issue resolved by help desk.")
    .UpdateAsync();
```

⚙️ TdxClientOptions

Customize retry/throttling behavior:

```
var options = new TdxClientOptions
{
    MaxRetries = 3,
    EnableThrottleCountdownLogging = true,
    OnThrottleWait = remaining => Console.WriteLine($"Waiting
{remaining.TotalSeconds}..."),
    OnThrottleContinue = () => Console.WriteLine("Continuing.")
};
```

🧱 Structure

```
TdxClient
├── People
│   └── Search
├── Tickets
│   ├── Create
│   └── [ticketId]
│       ├── Get
│       ├── SetStatus
│       └── AddFeedEntry
```

📦 Extending

You can add more fluent builders by extending BaseRequestBuilder:

```
public class AssetsRequestBuilder : BaseRequestBuilder
{
    public AssetsRequestBuilder(TdxBaseClient client)
        : base("assets", client) { }

    public Task<Asset> GetAssetAsync(string id) =>
        SendAsync<Asset>(CreateRequest(HttpMethod.Get, id));
}
```

🧪 Testing

You can mock the `TdxBaseClient` and `HttpClient` layers for unit testing. Also would recommend implementing your own httpclient purely to repoint to

---

🔐 Authentication

Supports Web Services Key-based authentication. If your API uses OAuth, you'll need to extend `TdxBaseClient` accordingly.

📄 License

MIT

🙋 Support / Contribution

Open an issue or submit a pull request — contributions welcome!

Getting Started

API Structure

APIs

Accounts

- Contains methods for working with accounts/departments.

Applications

- Contains methods for working with user applications.

Attachments

- Contains methods for working with attachments.

Attributes

- Contains methods for working with custom attributes.

Auth

- Contains methods for authentication.

Days Off

- Contains methods for working with days off.

Feed

- Contains methods for working with feed entries.

Groups

- Contains methods for working with groups within the TeamDynamix people database.

Industries

- Contains methods for working with account/department industries.

Integrations

- Contains methods for working with integrations.

Locations

- Contains methods for working with locations.

People

- Contains methods for working with users and other individuals within the TeamDynamix people database.

Time

- Contains methods for creating, reading, updating, and deleting time entries as well as methods for getting information about time accounts and time reports (weekly timesheets).

Type Categories

- Contains methods for managing type categories, which are used to group project and ticket types.

User Management

- Contains methods for performing bulk operations upon users and other individuals within the TeamDynamix people database.

Asset/Configuration Management

Asset Searches/Reports

- Contains methods for working with asset saved searches in an assets/CIs application.

Asset Statuses

- Contains methods for working with asset statuses.

Assets

- Contains methods for working with assets.

Configuration Item Searches/Reports

- Contains methods for working with configuration saved searches in an assets/CIs application.

Configuration Item Types

- Contains methods for working with configuration item types.

Configuration Items

- Contains methods for working with configuration items and their relationships.

Configuration Relationship Types

- Contains methods for working with configuration relationship types.

Contracts

- Contains methods for working with contracts.

Maintenance Windows

- Contains methods for working with maintenance windows.

Product Models

- Contains methods for working with asset product models.

Product Types

- Contains methods for working with asset product types.

Vendors

- Contains methods for working with asset vendors.

Projects

Briefcase Files

- Contains methods for working with briefcase files.

Briefcase Folders

- Contains methods for working with briefcase folders.

Issues

- Contains methods for working with issues.

Links

- Contains methods for working with links.

Plans

- Contains methods for working with project plans.

Project Templates

- Contains methods for working with project templates.

Project Types

- Contains methods for working with project types.

Projects

- Contains methods for working with projects.

Risks

- Contains methods for working with risks.

Tasks

- Contains methods for working with project tasks.

Reporting

Reports

- Contains methods for working with Report Builder reports.

Roles

Functional Roles

- Contains methods for working with functional roles.

Resource Pools

- Contains methods for working with resource pools.

Security Roles

- Contains methods for working with security roles.

Self-Service

Knowledge Base

- Contains methods for working with Knowledge Base articles and categories.

Project Requests

- Contains methods for working with project requests.

Service Catalog

- Contains methods for working with the service catalog.

Tickets

Blackout Windows

- Contains methods for working with blackout windows.

Impacts

- Contains methods for working with ticket impacts.

Maintenance Activities

- Contains methods for working with maintenance activities.

Priorities

- Contains methods for working with ticket priorities.

Sources

- Contains methods for working with ticket sources.

Ticket Searches/Reports

- Contains methods for working with ticket saved searches in a ticketing application.

Ticket Statuses

- Contains methods for working with ticket statuses.

Ticket Tasks

- Contains methods for working with ticket tasks.

Ticket Types

- Contains methods for working with ticket types.

Tickets

- Contains methods for working with tickets.

Urgencies

- Contains methods for working with ticket urgencies.