GitObjectDb Documentation

Thomas Caudal

Sep 22, 2023

Contents

1 Start **3**

2 Commit changes **5**
2.1 Direct update 5
2.2 Stage in the index, then commit 5

3 Branches **7**
3.1 Direct update 7

4 Comparing commits **9**

5 Node references **11**

6 Mergins, Rebasing, Cherry-picking **13**

7 Node versioning **15**

GitObjectDb is designed to simplify the configuration management versioning. It does so by removing the need for hand-coding the commands needed to interact with Git.

The Git repository is used as a pure database as the files containing the serialized copy of the objects are never fetched in the filesystem. GitObjectDb only uses the blob storage provided by Git.

Here's a simple example:

1. Add a reference to *GitObjectDb* NuGet package
2. Define your own repository data model:

```
[GitFolder("Applications")]
public class Application : Node
{
    public string Name { get; set; }

    public string Description { get; set; }
}
[GitFolder("Pages")]
public class Table : Node
{
    public string Name { get; set; }

    public string Description { get; set; }
}
```

See Getting Started for how to manipulate data.

CHAPTER 1

Start

Once the data model is created (see home page), you can initialize a new repository like this:

```
var serviceProvider = new ServiceCollection()
    .AddGitObjectDb()
    .AddGitObjectDbSystemTextJson()
    .AddSingleton(new ConventionBaseModelBuilder()
        .RegisterAssemblyTypes(Assembly.GetExecutingAssembly())
        .Build())
    .BuildServiceProvider();
var factory = serviceProvider.GetRequiredService<ConnectionFactory>();
var connection = factory(path);
```

In the example above, a Json serializer has been used. Alternative serializers exist, like Yaml (see GitObjectDb nuget packages.)

Note: Once a connection has been established, the repository can be queried / updated using the connection object.

CHAPTER 2

Commit changes

Changes to the repository can be done using a composable syntax. There are two ways to commit changes to the repositories:

2.1 Direct update

```
connection
    .Update("main", c => c.CreateOrUpdate(table, parent: application))
        .Commit("Added table.", author, committer);
```

Note: Within the Update(...) method, multiple transformations can be defined using nested calls.

2.2 Stage in the index, then commit

Previous method stores transformation in-memory then creates a commit. In the case where you need to store transformations made persistent to be committed later (like what you would do when using Git with your files), you can use the index method:

```
connection
    .GetIndex("main", c => c.CreateOrUpdate(table, parent: application))
        .Commit("Added table.", author, committer);
```

Note: Since GitObjectDb uses a bare repository, the internal Git index database file cannot be used. GitObjectDb uses its own independant dbindex file that has the advantage that multiple indices can be used simultaneously on different branches.

CHAPTER 3

Branches

Branches can be accessed to perform queries or to send commands. In the example below, you can see how to read & checkout different branches:

3.1 Direct update

```
connection
        .Update("main", c => c.CreateOrUpdate(table with { Description =␣
↪newDescription }))
        .Commit(new("Some message", signature, signature));
connection.Checkout("newBranch", "main~1");
connection
        .Update("main", c => c.CreateOrUpdate(table with { Name = newName }))
        .Commit(new("Another message", signature, signature));
```

CHAPTER 4

Comparing commits

Log can be analyzed to extract changes made to nodes and resources.

```
var comparison = connection.Compare("main~5", "main");
var nodeChanges = comparison.Modified.OfType<Change.NodeChange>();
```

CHAPTER 5

Node references

Node references allows linking existing nodes in a repository.

```
public record Order : Node
{
        public Client Client { get; set; }
        // ...
}
public record Client : Node
{
        // ...
}
// Nodes get loaded with their references (using a shared )
var cache = new Dictionary<DataPath, ITreeItem>();
var order = connection.GetNodes<Order>("main", referenceCache: cache).First();
Console.WriteLine(order.Client.Id);
```

CHAPTER 6

Mergins, Rebasing, Cherry-picking

Just like with Git, GitObjectDb let you do these different operations.

```
// main:      A---B    A---B
//             \    ->  \   \
// newBranch:   C        C---x

connection
        .Update("main", c => c.CreateOrUpdate(table with { Description =␣
↪newDescription }))
        .Commit(new("B", signature, signature));
connection.Repository.Branches.Add("newBranch", "main~1");
connection
        .Update("newBranch", c => c.CreateOrUpdate(table with { Name = newName }))
        .Commit(new("C", signature, signature));

sut.Merge(upstreamCommittish: "main");
```

CHAPTER 7

Node versioning

There can be scenario where you want to introduce changes to the code-first metadata that will change the way the data gets serialized. GitObjectDb let you define how the old nodes that have been stored in past commits can be adapted at runtime to the latest version:

```
[GitFolder(FolderName = "Items", UseNodeFolders = false)]
[IsDeprecatedNodeType(typeof(SomeNodeV2))]
private record SomeNodeV1 : Node
{
        public int Flags { get; set; }
}

[GitFolder(FolderName = "Items", UseNodeFolders = false)]
private record SomeNodeV2 : Node
{
        public BindingFlags TypedFlags { get; set; }
}
```

You then want to introduce a new change so that the *Flags* property contains more meaningful information, relying on enums:

```
[GitFolder(FolderName = "Items", UseNodeFolders = false)]
private record SomeNodeV2 : Node
{
        public BindingFlags TypedFlags { get; set; }
}
```

All you need to do is to #1 add the *[IsDeprecatedNodeType(typeof(SomeNodeV2))]* attribute. This will instruct the deserializer to convert nodes to new version, using a converter. #2 converter needs to be provided in the model. You can use AutoMapper or other tools at your convenience.

```
[GitFolder(FolderName = "Items", UseNodeFolders = false)]
[IsDeprecatedNodeType(typeof(SomeNodeV2))]
private record SomeNodeV1 : Node
{
```

(continues on next page)

(continued from previous page)

```
        // ...
}
var model = new ConventionBaseModelBuilder()
        .RegisterType<SomeNodeV1>()
        .RegisterType<SomeNodeV2>()
        .AddDeprecatedNodeUpdater(UpdateDeprecatedNode)
        .Build();
Node UpdateDeprecatedNode(Node old, Type targetType)
{
        var nodeV1 = (SomeNodeV1)old;
        return new SomeNodeV2
        {
                Id = old.Id,
                TypedFlags = (BindingFlags)nodeV1.Flags,
        };
}
```