When creating automated tests, we recommend using unit tests whenever possible. Unit tests are able to run without external resources, such as a database. They have the fastest execution speed (compared with other types of automated tests) and are the easiest to integrate into your development process.

A common problem with unit tests is that code needs to connect to a database, either directly or indirectly through external dependencies. Unit tests where this problem occurs throw the following exception upon execution:

"System.InvalidOperationException: The ConnectionString property has not been initialized"

In many cases, you can solve the problem by preparing faked data for your unit tests.

If you cannot fake the data required for a test, you need to create an integration or isolated integration test instead. Integration tests can work with a database, but have significantly slower execution speed. See:

- Creating integration tests with a connection string
- Creating isolated integration tests

Faking data

The Kentico API uses *Info* and *Provider* classes to manage the data stored in database tables. See the Database table API page for more information.

To create fake data for your unit tests, call the **Fake** method from the **UnitTests** base class (must be inherited by all unit test classes using the *CMS.Tests* library). The method prepares a faked instance of a specific Provider class with data. You can then use the given Provider's methods to **get**, **create** or **update** data in your unit tests.

Note: Faked providers do NOT allow you to use methods that **delete data**. If you need to test methods that delete data, use isolated integration tests.

Example (NUnit)

```
using CMS.Membership;
using CMS.Tests;
using NUnit.Framework;

[TestFixture]
public class MyUnitTests : UnitTests
{
        [SetUp]
        public void MyUnitTestSetUp()
        {
                // Prepares faked data for the UserInfoProvider
                Fake<UserInfo, UserInfoProvider>().WithData(
                        new UserInfo
                        {
                                UserID = 123,
                                UserName = "FakeUser",
                                UserNickName = "FakeUser"
                        });
        }

        [Test]
        public void MyTest()
        {
                // Calls a UserInfoProvider method to get user data
                var users = UserInfoProvider.GetUsers();
        }
}
```

In the example, *UserInfoProvider* is faked with specific data. Calling *UserInfoProvider* methods to get *UserInfo* objects within tests then returns the faked data instead of accessing the database. The code in the example behaves as if there were a single user named *FakeUser* in the database.

You do **not need to clean up the faked data before or after running tests**. Tests inherited from the **UnitTests** base class automatically reset all faked data upon initialization and cleanup.

Faking page data

Use the following approach if you need to fake page data in unit tests:

1. Generate a strongly-typed class for the page type that you wish to test.
2. Add the generated class into your test project.
3. In your test's *SetUp* method, register and fake the given page type by calling the following methods:
   - **DocumentGenerator.RegisterDocumentType<TGeneratedItemClass>**
   - **Fake().DocumentType<TGeneratedItemClass>**
4. In your test methods, create pages of the registered type and perform any required assertions.

Example (NUnit)

```
using CMS.DocumentEngine;
using CMS.DataEngine;
using CMS.Tests;
using Tests.DocumentEngine;

using NUnit.Framework;

// Namespace used by the 'CustomPageType' generated page type class
using CMS.DocumentEngine.Types.Custom;

[TestFixture]
public class MyUnitTests : UnitTests
{
        [SetUp]
    public void MyUnitTestSetUp()
    {
        // Registers the page type class
        // The 'CustomPageType' class is generated for a page type from the Kentico Page
types application
        DocumentGenerator.RegisterDocumentType<CustomPageType>(CustomPageType.CLASS_NAME);

        // Fakes the page type to allow creation of fake page data
        Fake().DocumentType<CustomPageType>(CustomPageType.CLASS_NAME);
    }

    [Test]
    public void MyTest()
    {
        // Creates a fake page for testing purposes
        var page = TreeNode.New(CustomPageType.CLASS_NAME).With(p =>
        {
            p.NodeAlias = "Test page";
            p.DocumentCulture = "en-US";
            p.SetValue("CustomTextField", "Test_Value");
        });

        // Tests the value of a custom field within the page type
        Assert.AreEqual("Test_Value", page.GetValue("CustomTextField"));
    }
}
```

Faking Info metadata

Calling the constructor of an *Info* class requires access to metadata stored in the database. If your unit tests do not require faked Providers with data prepared in advance, but directly create new instances of Info classes, you can fake only the Info metadata.

To prepare faked metadata for an Info class, call the **Fake** method with a single generic type parameter matching the given Info class.

Example (NUnit)

```
using CMS.Membership;
using CMS.Tests;
using NUnit.Framework;

[TestFixture]
public class MyUnitTests : UnitTests
{
        [SetUp]
        public void MyUnitTestSetUp()
        {
                // Sets up faked metadata for UserInfo
                Fake<UserInfo>();
        }

        [Test]
        public void MyTest()
        {
                // Creates a new UserInfo instance
                UserInfo user = new UserInfo();
        }
}
```