ACUMATICA

TestSDK Setup Guide

Complete Beginner's Onboarding Documentation

📚 Introduction to Acumatica TestSDK

Welcome to Acumatica TestSDK!
This comprehensive guide will help you understand and implement test automation for Acumatica ERP. By the end of this document, you'll have a complete understanding of installation, configuration, test creation, and execution.

What is Acumatica TestSDK?

Acumatica TestSDK is an automation framework provided by Acumatica that enables QA engineers, developers, and consultants to write UI automation tests for ERP screens using C#.

Key Benefits

🔄 Automation
Automate repetitive testing tasks and save valuable time

✅ Validation
Validate business logic and ensure data integrity

🔍 Regression Testing
Perform comprehensive regression testing efficiently

📸 Documentation
Capture detailed logs and screenshots for each test step

🚀 CI/CD Integration
Execute tests in CI/CD environments seamlessly

🎯 Screen Wrappers
Interact with Acumatica UI through auto-generated wrappers

💡 **How It Works:** TestSDK interacts with the Acumatica UI through **Screen Wrappers** that are generated using the Test Recorder or Wrapper Generator tools.

🛠️ Prerequisites

Before starting with Acumatica TestSDK, ensure you have the following:

- [ ] **Acumatica Instance** - Local or remote ERP installation
- [ ] **Visual Studio 2022+** - Latest version recommended
- [ ] **.NET Framework 4.8** - Required for TestSDK
- [ ] **TestSDK Package** - Provided by Acumatica (version-matched)
- [ ] **Chrome Browser** - For recording wrappers and test execution
- [ ] **Basic C# Knowledge** - Understanding of C# and NUnit framework

⚙️ Complete Setup Checklist (From Scratch)

Follow these steps carefully to configure your machine from zero and set up everything required to run Acumatica TestSDK successfully.

1 Install Acumatica ERP Instance

Install a local Acumatica ERP instance or connect to a hosted instance.

Note: Ensure your instance is running and accessible before proceeding.

2 Download Correct Version of Test SDK

Download the TestSDK version that exactly matches your Acumatica ERP build.

⚠️ **Important:** Version mismatch between TestSDK and Acumatica ERP can cause compatibility issues.

3 Install Visual Studio & Create Solution

- Install **Visual Studio 2022+**
- Create a new `Console App (.NET Core)` project
- Choose **.NET 6** or **.NET 7**

Project Setup: File → New → Project → Console App (.NET Core)

4 Add TestSDK NuGet Package Source

Configure NuGet to recognize TestSDK packages:

1. Go to `Tools → NuGet Package Manager → Package Manager Settings`
2. Navigate to `Package Sources`
3. Add new source pointing to TestSDK `/Packages` folder

5 Install Required NuGet Packages

Install the following packages for your solution:
`Tools → NuGet Package Manager → Manage NuGet Packages for Solution`

- [ ] **Core**
- [ ] **ClassGenerator**
- [ ] **Execution**
- [ ] **PX.QA.Tools**
- [ ] **PX.QA.WrappersGenerator**
- [ ] **GeneratedWrappers.Acumatica**

6 Configure Config.xml

Edit the `Config.xml` file in the parent directory:

- Set Chrome path in `<browserbin>`
- Set login credentials in `<login>` and `<pswd>`
- Set output log folder path in `<outputFolder>`
- Set test runner class in `<testing><Check Name="TestRunner"/>`

```
<configuration> <browserbin>C:\Program Files\Google\Chrome\Application\chrome.exe</browserbin> <login>admin</login>
<pswd>yourpassword</pswd> <outputFolder>C:\TestResults</outputFolder> <testing> <Check Name="TestRunner"/> </testing>
</configuration>
```

7 Add Config.xml Path in Project Properties

Configure your project to use the Config.xml file:

1. Right-click on your project in Solution Explorer
2. Select `Properties`
3. Navigate to `Debug` section
4. In `Command line arguments`, add the full path to Config.xml

Example: `C:\Projects\TestSDK\Config.xml`

8 Configure ClassGenerator.exe.config

Edit the `ClassGenerator.exe.config` file in the ClassGenerator directory:

- Set Instance Path in `<SitePhysicalPath>`
- Create a new folder named **'Out'** in your project directory
- Set the path to this folder in `<GenResultPath>`

```
<configuration> <appSettings> <add key="SitePhysicalPath" value="C:\Acumatica\Site"/> <add key="GenResultPath"
value="C:\Projects\TestSDK\Out"/> </appSettings> </configuration>
```

✅ **Setup Complete!** Your Acumatica TestSDK environment is now ready for test automation.

📁 TestSDK Project Structure

Understanding the project structure is crucial for effective test development. Here's a typical TestSDK project layout:

```
TestSDK
│
├── Program.cs              → Test runner entry point
├── TestAcumatica.csproj    → Project file
├── TestRunner.cs           → Executes all tests
├── Config.xml              → Configuration file
│
├── GeneratedWrappers/
│   └── Auto-generated screen wrappers
│
├── Tests/
│   ├── SalesOrderTest.cs
│   ├── CustomerTest.cs
│   └── InventoryTest.cs
│
├── Logs/
│   ├── HTML test results
│   ├── Screenshots/
│   └── JSON logs (optional)
│
└── Out/
    └── Generated wrapper output
```

Key Components Explained

Program.cs
Entry point for test execution. Initializes the test framework and starts the test runner.

TestRunner.cs
Orchestrates test execution, manages test lifecycle, and handles reporting.

GeneratedWrappers/
Contains auto-generated C# classes that represent Acumatica screens and their elements.

Tests/
Your test classes live here. Each file contains test cases for specific functionality.

Logs/
Test execution results, HTML reports, screenshots, and JSON logs are stored here.

Config.xml
Central configuration file for browser settings, credentials, and output paths.

💡 **Best Practice:** Keep your test files organized by module or functionality for easier maintenance and navigation.

🎯 Generating Screen Wrappers

Screen wrappers are C# classes that represent Acumatica screens. They provide a strongly-typed interface to interact with UI elements.

Why Use Screen Wrappers?

- **Type Safety:** Compile-time checking of screen elements
- **IntelliSense Support:** Auto-completion in Visual Studio
- **Maintainability:** Changes to screens are reflected in generated code
- **Readability:** Tests are easier to read and understand

Steps to Generate Wrappers

1 Edit ClassGenerator.exe.config

Open the configuration file in your ClassGenerator directory.

2 Set Screen Code

In `<FilenameFilter>`, specify the screen code you want to generate.

```
<add key="FilenameFilter" value="SO301000"/>
```

Example Screen Codes:

- **SO301000** - Sales Order Entry
- **AR303000** - Customer Entry
- **IN202500** - Inventory Item Entry

3 Save the Configuration

Save the ClassGenerator.exe.config file.

4 Generate Wrapper

Run `ClassGenerator.exe` to generate the screen wrapper.
The generated wrapper will appear in your **Out** folder, then copy it to **GeneratedWrappers** folder in your project.

✅ **Wrapper Generated!** You can now use the wrapper class in your test cases to interact with the Acumatica screen.

✍️ Writing Your First Test Case

Let's create a simple test that creates a Sales Order in Acumatica.

Basic Test Structure

```
using Core.Log; using Core.TestExecution; using GeneratedWrappers.Acumatica; using NUnit.Framework; namespace
Tests_Acumatica_demo.Tests { public class SalesOrder : Check { SO301000_SOOrderEntry screen = new SO301000_SOOrderEntry();
public override void Execute() { using (TestExecution.CreateTestCaseGroup("Create Sales Order")) { screen.OpenScreen();
Log.Screenshot(); screen.DocumentDetails.ServiceOrderNbr.Type("SO0001");
screen.DocumentDetails.Customer.Type("TESTCUSTOMER"); screen.Actions.Save.Click(); Log.Screenshot(); } } } }
```

Understanding the Code

Imports
`Core.Log` - For logging and screenshots
`Core.TestExecution` - Test execution framework
`GeneratedWrappers.Acumatica` - Screen wrappers

Test Class
Inherits from `Check` base class. Each test class represents a test scenario.

Screen Instance
Creates an instance of the generated wrapper for the Sales Order Entry screen.

Execute Method
Contains the actual test logic. Overrides the base `Execute()` method.

Test Case Group
`CreateTestCaseGroup` groups related actions for better reporting.

Screenshots
`Log.Screenshot()` captures the current screen state for documentation.

Common Test Actions

Opening a Screen

```
screen.OpenScreen();
```

Typing in Fields

```
screen.DocumentDetails.Customer.Type("CUSTOMER001"); screen.DocumentDetails.Description.Type("Test Order");
```

Clicking Buttons

```
screen.Actions.Save.Click(); screen.Actions.Release.Click();
```

Taking Screenshots

```
Log.Screenshot(); Log.Screenshot("After saving order");
```

Validating Data

```
Assert.AreEqual("Open", screen.DocumentDetails.Status.GetValue());
Assert.IsTrue(screen.DocumentDetails.OrderTotal.GetValue() > 0);
```

💡 **Best Practices:**

- Use meaningful test case names
- Take screenshots at key points in the test
- Add assertions to validate expected outcomes
- Group related actions using TestCaseGroup
- Keep tests independent and reusable

📊 Test Results & Reporting

After test execution, TestSDK generates comprehensive HTML reports with detailed information about each test run.

What's Included in Reports?

📝 Test Summary
Overview of all tests executed with pass/fail counts

📸 Screenshots
Visual evidence of every logged action

⏱️ Execution Time
Duration of each test and step

✅ Pass/Fail Status
Clear indication of test outcomes

🐛 Error Details
Stack traces and error messages for failures

📋 Step-by-Step Log
Detailed log of every action performed

Report Location

```
/Logs/<date-time>/
├── report.html          → Main HTML report
├── Screenshots/
│   ├── step_001.png
│   ├── step_002.png
│   └── ...
└── logs.json            → Optional JSON logs
```

Understanding Report Structure

Test Name

The name of your test class and method

Steps Executed

Chronological list of all actions performed during the test

Screenshots

Visual snapshots taken at each `Log.Screenshot()` call

Pass/Fail Status

Clear visual indicator showing whether the test succeeded or failed

Error Stack Trace

When a test fails, detailed error information including line numbers and exception details

💡 **Pro Tip:** Review the HTML reports after each test run to verify that your tests are executing as expected. The screenshots provide valuable visual feedback.

🎉 **Congratulations!** You now have a complete understanding of Acumatica TestSDK. You're ready to start automating your Acumatica ERP testing!

🚀 Next Steps

1. Practice
Create test cases for different Acumatica screens you use regularly

2. Organize
Structure your tests into logical groups by module or functionality

3. Automate
Set up automated test execution in your CI/CD pipeline

4. Expand
Generate wrappers for additional screens as needed

5. Maintain
Keep your tests and wrappers updated with Acumatica changes

6. Share
Document your test cases and share knowledge with your team

Back to Top

ACUMATICA

Acumatica TestSDK - Beginner's Guide

Complete documentation for setting up and using Acumatica TestSDK for automated testing

Chetan Salunkhe | QA, Zingworks LLP