Tyche – Technical Architecture

Table of Contents

[1 What is Tyche 1](#_Toc312288115)

[1.1 Tyche in simple words 1](#_Toc312288116)

[1.2 Tyche Components 2](#_Toc312288117)

[2 Tyche Architecture (POC) 2](#_Toc312288118)

[2.1 Desktop Client 2](#_Toc312288119)

[2.2 Desktop Client in simple words 2](#_Toc312288120)

[3 Compilation 4](#_Toc312288121)

[3.1 Projects that compile 4](#_Toc312288122)

[3.2 MSBuild “AfterBuild” target 5](#_Toc312288123)

[3.3 How to compile 6](#_Toc312288124)

[4 How things are organized at the code level 7](#_Toc312288125)

[4.1 Three projects one unit 7](#_Toc312288126)

# What is Tyche

## Tyche in simple words

Tyche is an intelligence platform for decision making purposes in the trading industry.

## Tyche Components

Tyche exposes the following layers for both end users and content providers:

* TycheCore © - The core logic of the intelligence produced by Tyche
* TycheApi © - Client API, used by client applications
* TycheApi © - Integration API, used by third party content providers to integrate into Tyche
* TycheDesk ©– Client application for Desktop
* TycheTab ©- Client application for Tablets & Mobiles

# Tyche Architecture (POC)

## Desktop Client

**JS Proxy Objects**

**Script Factory**

**WCF**

**IE 9 Runtime**

**Windows App (Win Forms / WPF)**

**HTML 5 + CSS**

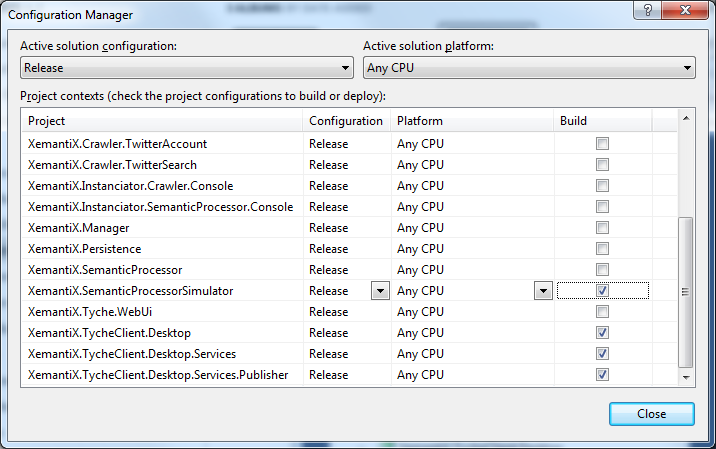
## Desktop Client in simple words

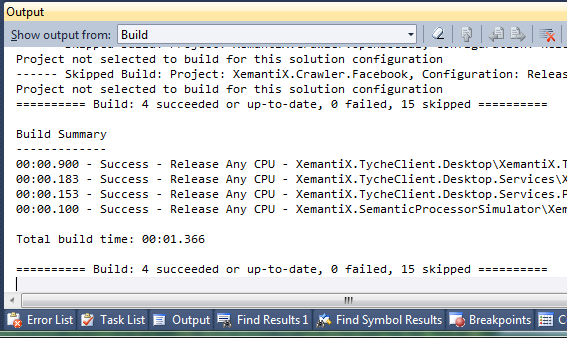
The idea behind this technique is to achieve for a windows desktop application an UI rendered by the HTML5 +CSS3 and allow the business logic to be written in high performance an high efficiency language as C#.

As a minimal requirement this development model for this kind of applications must have a MVC at its core to ease thing in long term. Thus the Controller is the JS part that serves as a bridge between the Model (C#) and the View (HTML5+CSS3). The model and its data is exposed as DataContracts to the JS by passing through the Scripting Factories existing in .NET. All is possible in a desktop app because in .NET exists the System.Web.Hosting namespace. The same namespace will eventually allow to replace the current IIS Express based implementation to serve WCF services to the JS.

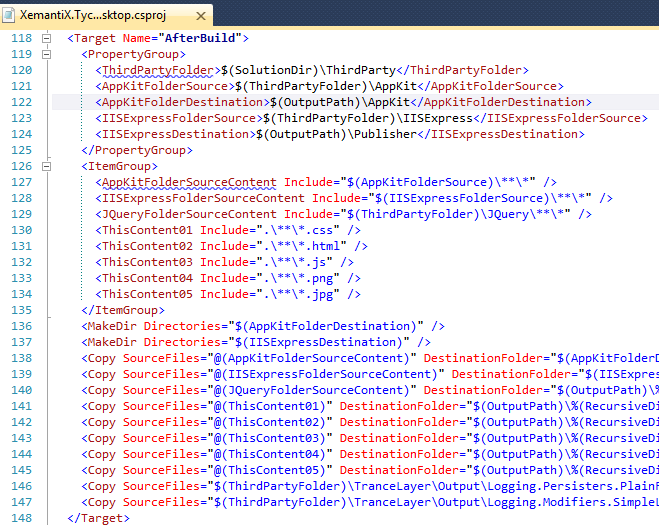
# Compilation

## Projects that compile





## MSBuild “AfterBuild” target

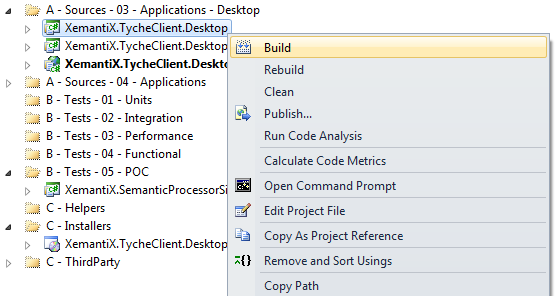


This will build & deploy to the “$(OutputPath)” a part of the files that are needed to run the app

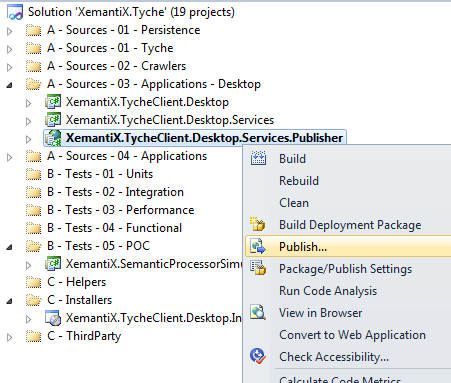
## How to compile

In order to compile the project you have to do the following steps:

1. Build the “XemantiX.TycheClient.Desktop”



1. Publish the “XemantiX.TycheClient.Desktop.Services.Publisher”



# How things are organized at the code level

## Three projects one unit

For the POC and to speed things up both at development and abstraction levels for now the “to be” a single unit is split into 3 projects:

1. “XemantiX.TycheClient.Desktop”

This is the core app that binds all things together. The Windows components initialization part + HTML5 components + and communication channels.

1. “XemantiX.TycheClient.Desktop.Services”

The part of project that defines the WCF services that are called from JS and exchange data in JSON format between the Controller and the View.

1. “XemantiX.TycheClient.Desktop.Services.Publisher”

This component used to server as a bridge between the JS and C# logic. Currently it is based on the IIS Express. During the time that the research team will be working on the true algorithms for Tyche’s logic this component will be replaced and merged into the “XemantiX.TycheClient.Desktop” and will replace the current content of the “AspNetServer.cs”.

1. “XemantiX.SemanticProcessorSimulator”

This will simulate the semantic processor so that we can achieve the following:

* Make room so that research team can work on algorithms without disturbing the TycheDesk development
* Isolate the App’s dependency on input data so it can have data as much as it needs in order to have a proper development
* Show the data at the UI level as specified the business requirements docs