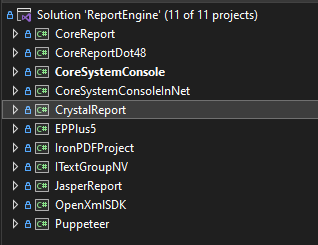
Table of Contents

[1 Solution Folders and structure 1](#_Toc117608692)

# Solution Folders and Structure



## Startup project

Because Crystal Report only support under .net framework, you must change the start-up project to “CoreSystemConsoleInNet” for debug, and test Crystal Report.

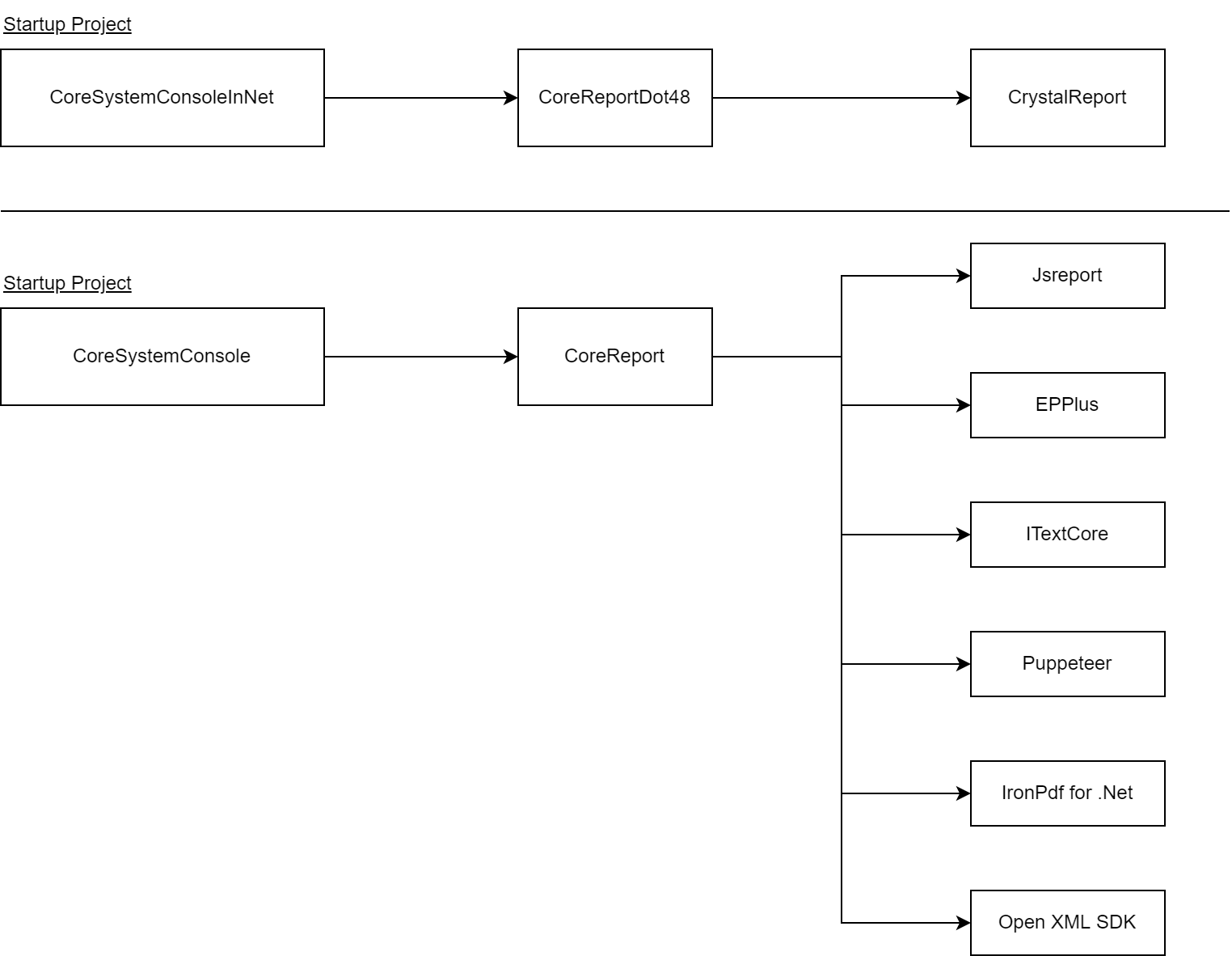
For the future the project would create in .Net Core probably (5 is the latest when I create this github repository, 6 is publish when I create this document), that is why I choose .Net Core in other projects.

**For Crystal Report, change startup project to “CoreSystemConsoleInNet”**

Program flow: CoreSystemConsoleInNet (.net framework 4.8) 🡺 CoreReportDot48 🡺 CrystalReport

**For other report library, change startup project to “CoreSystemConsole”**

CoreSystemConsole (.Net Core 5) 🡺CoreReport 🡺 JSReport / EPPlus / ITextCore / Puppeteer / IronPdf for .Net / Open Xml SDK (MS office sdk)



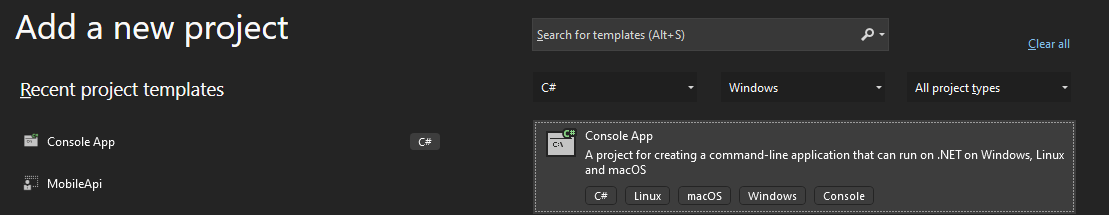
# Add a new project

Steps

1. Add new project
2. Create structured folders
3. Create report engine Decorator
4. Create report entity
5. Create program entity
6. Update CoreSystemConsole main program

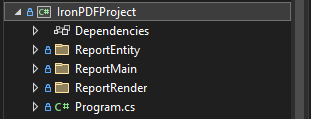
## Add new project

To use a new report library, create a new Console App project.



## Create structured folders

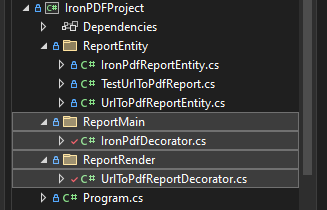
Create folders as below structure



## Create report engine Decorator

In ReportMain folder, create a generic Decorator that inheritance VisualizationDecorator

For IronPDFProject, I created IronPdfDecorator.cs, you should put all common object and/or initialize variable that would be used by the report engine in this class.



For IronPDF, it could generate pdf in several approaches, include

* HTML(source code) to PDF
* HTML Files to PDF
* URL to PDF
* ASPX Pages to PDF
* Images To PDF
* TIFF to PDF with Multi-Page Support
* Javascript in Html To Pdf

To separate the purpose and related resource in terms of different approach, I also created UrlToPdfReportDecorator.cs to control the data flow of “URL to PDF”. UrlToPdfReportDecorator.cs is inheritance IronPdfDecorator.cs

## Create report entity

In ReportEntity folder, create a generic Report Entity that inheritance BaseReportEntity

For IronPDFProject, I created IronPdfReportEntity.cs,

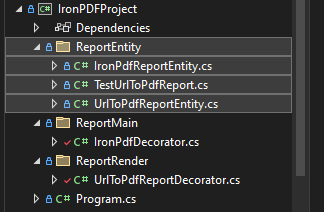
UrlToPdfReportEntity inheritance IronPdfReportEntity.cs

To separate the purpose and related resource in terms of different approach, there should be

* HtmlSourceToPdfReportEntity
* HtmlFilesToPdfReportEntity
* ImageToPdfReportEntity
* …etc

To separate the purpose and related resource in terms of different approach, all report programs that should inheritance UrlToPdfReportEntity if it is going convert a URL to PDF. For me to test the URL to PDF function, I also created TestUrlToPdfReport.cs to control the data of “URL to PDF”.

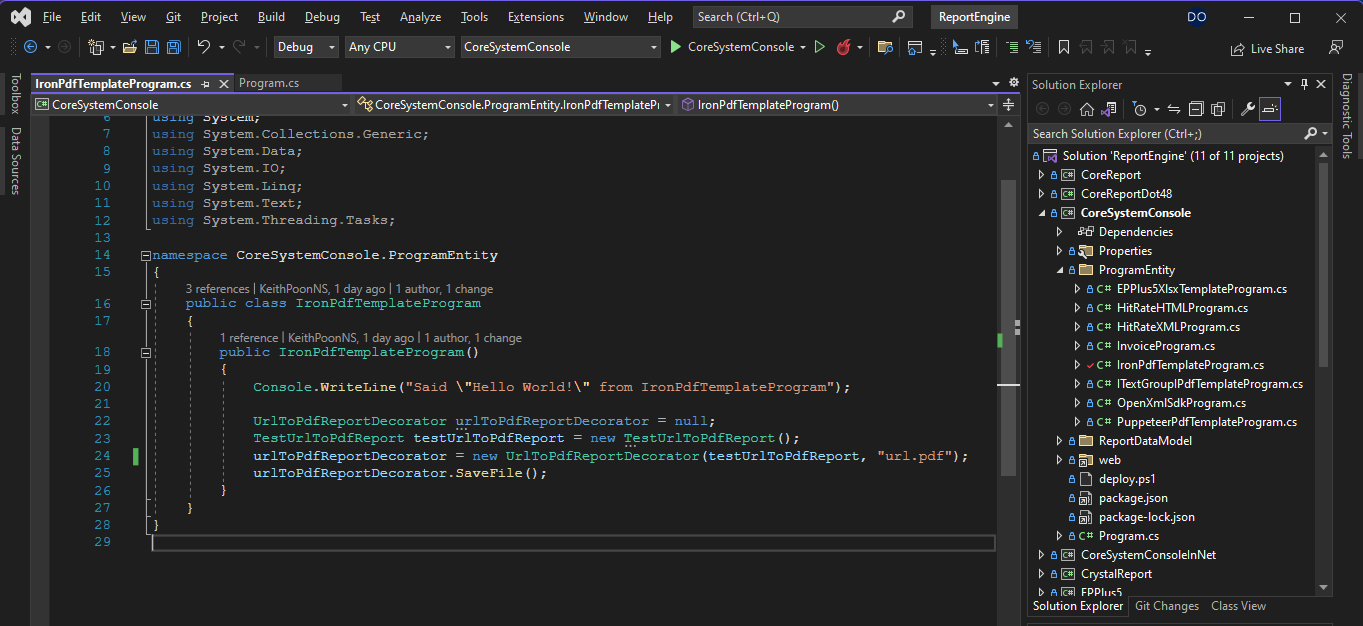
You could define and overwrite the “Template Directory”, “Template File”, Header Setting, Footer Setting, Data set here



## Create program entity

In CoreSystemConsole project, ProgramEntity folder, create a program that will initialize the report instance and report decorator

I created IronPdfTemplateProgram.cs to test the above Iron programs.



## Update CoreSystemConsole main program

In the root of CoreSystemConsole project, open and update program.cs

Add and call IronPdfTemplateProgram

IronPdfTemplateProgram ironPdf = new IronPdfTemplateProgram();

