Documentation

Everything you need to integrate SSRS2 into your .NET application.

1 Quick Start

Get up and running with SSRS2 in under 5 minutes.

1. Install the NuGet package

```
$ dotnet add package SSRS2.NETCore
```

2. Add the using statement

```
using SSRS2.Reporting.NETCore;
```

3. Generate your first PDF

```
// Load your RDLC report
using var report = new LocalReport();
report.LoadReportDefinition(File.OpenRead("Report.rdlc"));

// Add your data
report.DataSources.Add(
    new ReportDataSource("DataSetName", yourData));

// Render to PDF
var pdfBytes = report.Render("PDF");

// Save or return the PDF
File.WriteAllBytes("output.pdf", pdfBytes);
```

2 Installation

.NET CLI

```
dotnet add package SSRS2.NETCore
```

Package Manager Console

```
Install-Package SSRS2.NETCore
```

PackageReference (csproj)

```
<PackageReference Include="SSRS2.NETCore" Version="1.7.0" />
```

Supported frameworks: .NET 8.0, .NET 9.0, .NET 10.0, .NET 11.0, and .NET Framework 4.7.2 / 4.8 — so you can adopt SSRS2 in place, without first porting your existing Windows application.

The 2.0 release candidate: SSRS2.NETCore 2.0.0-rc.2 is a prerelease of the engine the Node.js and Python packages run on (`dotnet add package SSRS2.NETCore --prerelease`). 1.7.0 is the stable version.

From Node.js, Python, Java or any language

Render your existing RDLC reports from Node.js, Python and Java, as Report Viewer renders them: charts, gauges, maps and all. No conversion, no SSRS, no Windows. Each package carries the engine and the runtime it needs, so no .NET is installed; calls run in worker processes the package starts and keeps warm. They are prereleases of 2.0,

covered by the same licence key.

.NET `dotnet add package SSRS2.NETCore` NuGet 1.7.0

.NET Framework 4.7.2 and 4.8, and .NET 8, 9, 10 and 11 on Windows, Linux and macOS

Node.js `npm install @ssrs2/ssrs2` npm 2.0.0-rc.2

Node.js 22 or later, on Linux with glibc (x64, Arm64), macOS (Arm64, x64) and Windows (x64, Arm64)

Python `pip install --pre ssrs2` PyPI 2.0.0rc2

Python 3.9 or later, on Linux with glibc (x64, Arm64), macOS (Arm64, x64) and Windows (x64, Arm64)

Java `net.ssrs2:ssrs2:2.0.0-rc.2` Maven 2.0.0-rc.2

Java 17 or later, on Linux with glibc (x64, Arm64), macOS (Arm64, x64) and Windows (x64, Arm64)

HTTP

```
docker run -p 8080:8080 ghcr.io/antoinegou/ssrs2-server:2.0.0-rc.2
```

Docker, linux/amd64 and linux/arm64

Node.js

```
// npm install @ssrs2/ssrs2
const fs = require('node:fs/promises');
const { renderReport, checkReport, markdownTo, importReport } =
require('@ssrs2/ssrs2');

const pdf = await renderReport({ path: 'Reports/Invoice.rdlc' }, {
  format: 'PDF',                // or WORDOPENXML, EXCELOPENXML, PPTX,
IMAGE, PNG, HTML5, CSV, XML
  datasets: { Items: [{ Description: 'Widget', Price: 104.99, Qty: 1 }] },
  parameters: { InvoiceNumber: '42' },
});
await fs.writeFile('invoice.pdf', pdf.file);

const check = await checkReport({ path: 'Reports/Invoice.rdlc' });   // a
report's problems without rendering it
const deck = await markdownTo({ path: 'talk.md' }, 'pptx');          //
Markdown to a PowerPoint deck
const rdlc = await importReport({ path: 'Invoice.frx' });            //
another tool's report as an RDLC
```

A report is its text, its bytes or `{ path }`. A render answers `{ file, contentType, extension, pages, notes, elapsedMs, trial }`. @ssrs2/ssrs2 runs on weblitzpdf, the same engine at the same version: npm installs the engine for your platform only, as one of its optional packages. Workers, timeouts, errors and a server's settings: weblitzpdf.com/docs/node.

Python

```
# pip install --pre ssrs2
from pathlib import Path
import ssrs2

rows = [{"Description": "Widget", "Price": 104.99, "Qty": 1}]
ssrs2.render_report(Path("Reports/Invoice.rdlc"), datasets={"Items": rows},
path="invoice.pdf")
ssrs2.render_report(Path("Reports/Invoice.rdlc"), format="docx", datasets=
{"Items": dataframe})

problems = ssrs2.check_report(Path("Reports/Invoice.rdlc"))
deck = ssrs2.markdown_to(Path("talk.md"), "pptx")
rdlc = ssrs2.import_report(Path("Invoice.frx"))
```

A report is its XML as a `str`, its bytes or a `pathlib.Path`; a dataset's rows a list of dicts or a pandas or polars DataFrame. ssrs2 runs on weblitz, the same engine. PyPI's version is 2.0.0rc2, Python's spelling of 2.0.0-rc.2, the version of every package. More: weblitzpdf.com/docs/python.

Java

```
// net.ssrs2:ssrs2:2.0.0-rc.2, from https://maven.weblitzpdf.com
import com.weblitzpdf.*;
import net.ssrs2.Ssrs2;

record Item(String description, BigDecimal price, int qty) {}

Ssrs2.renderReport(Path.of("Reports/Invoice.rdlc"), new ReportOptions()
        .setFormat("PDF")   // or WORDOPENXML, EXCELOPENXML, PPTX, IMAGE,
PNG, HTML5, CSV, XML
        .setDataset("Items", List.of(new Item("Widget", new
BigDecimal("104.99"), 1)))
        .setParameter("InvoiceNumber", "42"))
     .save(Path.of("invoice.pdf"));

CheckResult check = Ssrs2.checkReport(Path.of("Reports/Invoice.rdlc"));
Result deck = Ssrs2.markdownTo(Path.of("talk.md"), "pptx");
Result rdlc = Ssrs2.importReport(Path.of("Invoice.frx"));
```

A report is a `Path`, its text or a `Source` (bytes, a stream); a dataset's rows records, beans, maps, lists or streams. net.ssrs2:ssrs2 runs on com.weblitzpdf:weblitzpdf, the same engine at the same version. Both are in Weblitz's own Maven repository, `https://maven.weblitzpdf.com`, which the build names; Maven adds the engine for your platform. The repository, Gradle, workers and a server's settings: weblitzpdf.com/docs/java.

Any language, over HTTP

```
docker run -p 8080:8080 -e SSRS2_LICENSE ghcr.io/antoinegou/ssrs2-
server:2.0.0-rc.2

# a report with its rows: the request as a field, the definition as a file
curl -F 'request={"format": "PDF", "datasets": {"Items": [{"Description":
"Widget", "Price": 9.5, "Qty": 2}]}}' \
     -F report=@Invoice.rdlc http://localhost:8080/v1/report -o invoice.pdf

# a report's problems, Markdown to a deck, another tool's report to RDLC
curl -F report=@Invoice.rdlc http://localhost:8080/v1/check
curl -F 'request={"format": "PPTX", "layout": "slides"}' -F deck=@deck.md
http://localhost:8080/v1/markdown -o deck.pptx
curl -F file=@Orders.frx http://localhost:8080/v1/import -o Orders.rdlc
```

The image answers /v1/report, /v1/check, /v1/markdown, /v1/import on port 8080 (`ssrs2-weblitz-server` adds HTML to PDF), for linux/amd64 and linux/arm64, with the fonts for every script. `SSRS2_SERVER_TOKEN` asks callers for a bearer token. Its endpoints, settings and limits: weblitzpdf.com/docs/server.

3 Basic Usage

Loading Reports

There are multiple ways to load an RDLC report:

```
// From file path
report.LoadReportDefinition(File.OpenRead("path/to/report.rdlc"));

// From embedded resource
var assembly = Assembly.GetExecutingAssembly();
var stream =
assembly.GetManifestResourceStream("MyApp.Reports.Report.rdlc");
report.LoadReportDefinition(stream);

// From byte array
var bytes = File.ReadAllBytes("report.rdlc");
report.LoadReportDefinition(new MemoryStream(bytes));
```

Adding Data Sources

```
// Single data source
var items = new List<Product> { ... };
report.DataSources.Add(
    new ReportDataSource("Products", items));

// Multiple data sources
report.DataSources.Add(new ReportDataSource("Orders", orders));
report.DataSources.Add(new ReportDataSource("Customers", customers));
```

Setting Parameters

```
report.SetParameters(new[] {
    new ReportParameter("ReportTitle", "Sales Report 2025"),
    new ReportParameter("ShowDetails", "true"),
    new ReportParameter("StartDate", DateTime.Now.ToString("yyyy-MM-dd"))
});
```

Enabling PDF Hyperlinks

```
// Enable hyperlinks in PDF output
report.EnableHyperlinks = true;

var pdf = report.Render("PDF");
```

Identical output on every platform

Report engines usually ask the operating system for the fonts a report requests. When the machine does not have one, each platform quietly substitutes something different — and the substitutes are not the same width. *Segoe UI* becomes Helvetica on macOS and DejaVu Sans on Linux, which is about 10% wider. Text then wraps differently, rows grow, and a table row that fits on your laptop falls off the page in your container.

SSRS2 ships metric-compatible fonts inside the package and renders from those, so the same RDLC lays out the same way on Windows, macOS, Linux and in Docker. Nothing to install on the host, and no `libgdiplus`.

Covered families: Arial, Helvetica, Segoe UI, Tahoma, Verdana, Trebuchet MS, Times New Roman, Georgia, Courier New, Consolas, Cambria, Calibri, Aptos and Aptos Narrow. Anything else falls back to the host's fonts, unless you register its files (below).

Checking it at startup

If the bundled fonts cannot be found, reports still render — they just stop being reproducible across machines. Assert it once at startup rather than discovering it from a mismatched PDF.

```
using SSRS2.Reporting.NETCore;

// false means the font files did not reach your output directory
if (!ReportFonts.BundledFontsAvailable)
    throw new InvalidOperationException("SSRS2 fonts missing — output will
vary by platform");

// Opt out and use the host's fonts instead (output becomes platform-
dependent)
ReportFonts.UseSystemFonts = true;
```

Your own fonts

A family SSRS2 does not ship, such as your corporate font, comes from each machine's fonts. Register its files once at startup, one per style, and it renders the same everywhere, ahead of the bundled faces. Cambria is drawn with Caladea set to Cambria's widths and line heights, so text breaks where it does in Report Viewer; register Cambria's own files, where their licence allows, for its own letter shapes.

```
ReportFonts.AddFontFile("fonts/Corporate-Regular.ttf");
ReportFonts.AddFontFile("fonts/Corporate-Bold.ttf");
```

4 Export Formats

PDF

```
var pdf = report.Render("PDF");
File.WriteAllBytes("report.pdf", pdf);
```

PDF: title, passwords, PDF/A

A PDF carries the report's display name as its title and the report's author, as Report Viewer's do; the device information sets them, a subject and keywords. With a password the PDF is encrypted (AES-256), and `PermitPrinting`, `PermitCopying` and `PermitModifying` say what its reader may do. `PdfA` writes PDF/A-2b, for archives.

```
report.DisplayName = "Sales by region";
var pdf = report.Render("PDF", "<DeviceInfo><Author>Finance</Author>"
    + "<OwnerPassword>to change it</OwnerPassword>
<PermitCopying>false</PermitCopying></DeviceInfo>");

// For archives: fonts embedded, the colour profile, what the file
says of itself
var archive = report.Render("PDF", "<DeviceInfo><PdfA>true</PdfA>
</DeviceInfo>");
```

PNG, every page (with custom DPI)

```
var deviceInfo = "<DeviceInfo><OutputFormat>PNG</OutputFormat>
<DpiX>300</DpiX><DpiY>300</DpiY></DeviceInfo>";

// IMAGE writes every page, a stream per page, as Report Viewer does
("PNG" alone renders StartPage)
var pages = new List<MemoryStream>();
report.Render("IMAGE", deviceInfo,
    (name, ext, enc, mime, seek) => {
        var stream = new MemoryStream();
        pages.Add(stream);
        return stream;
    }, out _);

for (var i = 0; i < pages.Count; i++)
    File.WriteAllBytes($"page{i + 1}.png", pages[i].ToArray());
```

JPEG (with quality control)

```
var deviceInfo = "<DeviceInfo><DPI>150</DPI><Quality>85</Quality>
</DeviceInfo>";

var jpeg = report.Render("JPEG", deviceInfo);
```

Word & Excel

```
// Word document
var docx = report.Render("WORDOPENXML");
File.WriteAllBytes("report.docx", docx);

// Excel spreadsheet
var xlsx = report.Render("EXCELOPENXML");
File.WriteAllBytes("report.xlsx", xlsx);
```

Format Name	Extension	MIME Type
PDF	.pdf	application/pdf
EXCELOPENXML	.xlsx	application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
WORDOPENXML	.docx	application/vnd.openxmlformats-officedocument.wordprocessingml.document
PPTX	.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation
HTML5	.html	text/html
HTML4.0	.html	text/html
MHTML	.mhtml	multipart/related
PNG	.png	image/png
JPEG	.jpg	image/jpeg
WEBP	.webp	image/webp
IMAGE	.tif	image/TIF (every page in one TIFF; with OutputFormat PNG or JPEG, a stream per page)
CSV	.csv	text/csv
XML	.xml	text/xml

Checking reports, and reports you did not write

Checking a report before it runs

`CheckDefinition()` compiles the report definition as rendering it would and returns each problem as a `Warning`: its code, the item and the property it is about, the message, and the expression when one does not compile. Nothing is rendered, no data is needed, and it does not throw. Useful before deploying a report, and wherever reports are generated or edited by a program.

```
var report = new LocalReport();
report.LoadReportDefinition(reportDefinition);

foreach (var problem in report.CheckDefinition())
    Console.WriteLine($"{problem.Severity} {problem.Code}
{problem.ObjectName}.{problem.PropertyName}: {problem.Message}");
// Error rsFieldReference Title.Value: The Value expression for the text box
'Title' refers to the field 'Nmae'...
```

Reports your application did not write

A report's expressions and custom code run with your application's permissions. For reports that come from elsewhere (uploaded by your users, written by a language model, converted from another tool), `RestrictedCode` holds them to the report object model, Visual Basic's functions and .NET's text, number, date, collection, regular expression and culture types. Files, the environment, processes, threads, reflection, native libraries and code modules are refused when the report is compiled, each error at its expression. It guards against mistakes and injected code; it is not a sandbox, and it does not limit time or memory: render hostile reports in a process of their own. On .NET 8, 9, 10 and 11.

```
var report = new LocalReport { RestrictedCode = true };
report.LoadReportDefinition(uploaded);
// =System.IO.File.ReadAllText("/etc/hosts") is an error:
// [SSRS2001] 'System.IO.File' is not available to a restricted report
var problems = report.CheckDefinition();
```

A viewer of your own

Rendered as HTML5 with the `ActionScript` device setting, a report calls your script function on each click: a group's rows opened, a column sorted, a bookmark followed, a drillthrough. `Toggle`, `Sort`, `FindBookmark`, `FindDocumentMapNode` and `Drillthrough` do what that click does in

Report Viewer; render the report again to show it.

```
var html = report.Render("HTML5", "<DeviceInfo>
<ActionScript>viewerAction</ActionScript></DeviceInfo>");

// The page calls viewerAction('Toggle', id, event): send the id to your
application
report.Toggle(id);
html = report.Render("HTML5", deviceInfo);
```

AI coding agents: the SSRS2 MCP server

An agent that writes or changes your `.rdlc` files (Claude Code, Codex, Cursor, VS Code, Visual Studio) can read, change and check them with SSRS2. `describe_report` outlines a report with the role of each item (table, totals, declaration, signature…); `compare_reports` groups many reports by their structure; `edit_report` changes one file or hundreds by short operations, leaving what it does not touch byte for byte, and says only the problems the change brought and the pages before and after; `validate_report` compiles the report as the engine would and returns each problem with its code, item, expression and a hint, and checks where items print, placement rules and row counts against a git baseline; `render_preview` returns its pages as images and what they print as text; `convert_report` writes PDF, Word, Excel or PowerPoint. The server runs on your machine, over stdio: your reports and data stay there, and a report's code runs restricted, in a process of its own. It needs the .NET 10 SDK.

```
# Once: downloads the tool (65 MB), so your client's first start is quick
dnx SSRS2.Mcp --yes -- --help

# Claude Code
claude mcp add ssrs2 -- dnx SSRS2.Mcp --yes

# Codex
codex mcp add ssrs2 -- dnx SSRS2.Mcp --yes

# .mcp.json (Claude Code), mcp.json (VS Code, Cursor)
{ "mcpServers": { "ssrs2": { "command": "dnx", "args": ["SSRS2.Mcp", "--
yes"] } } }

# A migration audit of a folder of reports: each one compiled, run and
graded
dnx SSRS2.Mcp --yes -- audit ./Reports --out ./ssrs2-audit
```

Or with nothing to install: the hosted server at `https://ssrs2.net/mcp` has the same tools, given the report or the Markdown as text (`rdlc` , or `markdown` with its `pictures`), and `convert_report` answers the file as a download for ten minutes. Calls are limited by address; a licence key in the header `Authorization: Bearer` raises the limit and drops the watermark. Nothing a call sends is kept after it (privacy).

```
# Claude Code
claude mcp add --transport http ssrs2 https://ssrs2.net/mcp

# Codex: ~/.codex/config.toml, the licence key in SSRS2_LICENSE (optional)
[mcp_servers.ssrs2]
url = "https://ssrs2.net/mcp"
bearer_token_env_var = "SSRS2_LICENSE"
```

In claude.ai, Claude Desktop or ChatGPT, which send no header of their own, add `https://ssrs2.net/mcp/licensed` as a custom connector: it asks you to sign in on ssrs2.net with your licence key, once, and then runs licensed (in Claude Code or Codex: add it, then `claude mcp login ssrs2` or `codex mcp login ssrs2`) (how an agent authenticates).

Without a licence key the tools work in full and previews carry a trial watermark. Setup for other clients: SSRS2.Mcp on NuGet. Two sessions of an agent with these tools, step by step: Designing SSRS reports with an AI agent.

An agent in the online designer

In the online designer, **Connect an agent** gives your browser an address for your agent: the agent then reads and changes the report open in your page while you watch the canvas and the preview, and each of its changes is one step you can undo. The address is your browser's, the same each time, and reaches a page only while you keep it connected.

```
# Claude Code
claude mcp add --transport http ssrs2-designer
https://designer.ssrs2.net/mcp/<your address>

# .mcp.json (Claude Code), mcp.json (VS Code, Cursor)
{ "mcpServers": { "ssrs2-designer": { "type": "http", "url":
"https://designer.ssrs2.net/mcp/<your address>" } } }
```

- `upload_report` opens a report file from the agent's machine (an .rdlc, an .rdl, or another tool's report the designer converts) with one command of its shell: the file never passes through the conversation, and the agent gets the report's outline and problems back.

- `review_report` has the designer's assistant review the report and answers its remarks in the call, each with its item, whether the agent's change brought it, and the fix; `edit_report` with `review: true` does it with a change. The assistant reviews for a signed-in person, or for a licence key the agent gives (`"headers": { "Authorization": "Bearer SSRS2-…" }` beside the url), with which it also reviews report files the agent changes on your machine, sent from its shell as upload_report sends them.

5 Docker Deployment

SSRS2 works seamlessly in Docker containers. Here's a minimal Dockerfile:

```
# Build stage
FROM mcr.microsoft.com/dotnet/sdk:10.0 AS build
WORKDIR /src
COPY . .
RUN dotnet publish -c Release -o /app

# Runtime stage
FROM mcr.microsoft.com/dotnet/aspnet:10.0
WORKDIR /app
COPY --from=build /app .

# On Linux, SSRS2 needs fontconfig, which the official .NET images lack
RUN apt-get update && apt-get install -y --no-install-recommends \
    libfontconfig1 \
    && rm -rf /var/lib/apt/lists/*

ENTRYPOINT ["dotnet", "YourApp.dll"]
```

Fonts: SSRS2 bundles fonts with the widths of the families listed above, so reports lay out alike on every platform; for other fonts, register their files with `ReportFonts.AddFontFile`.

6 Migration from Microsoft.ReportViewer

Migrating from Microsoft.ReportViewer is straightforward. Here are the key changes. For a complete comparison of migration options (including Power BI and cost analysis), see the SSRS to .NET Core migration guide.

1. Update Namespaces

Before (Microsoft):

```
using Microsoft.Reporting.WinForms;
// or
using Microsoft.Reporting.WebForms;
```

After (SSRS2):

```
using SSRS2.Reporting.NETCore;
```

2. API Compatibility

The core API is nearly identical:

- ✓ `LocalReport` class works the same way
- ✓ `Render()` method signature is compatible
- ✓ `ReportDataSource` and `ReportParameter` work identically
- ✓ Export format strings are the same ("PDF", "EXCELOPENXML", etc.)

3. RDLC Files - No Changes Needed!

Your existing .rdlc files work as-is. No conversion or modification required.

Related Resources

⚡ Full Feature List

s, gauges, PDF hyperlinks, images at any DPI you set and more.

🔄 Migration Guide

are SSRS, Power BI, and SSRS2 with cost analysis.

💰 Pricing

e licence per company, USD 799 a year, everything included.

Need a Hand?

Basic support is included with the licence: open a ticket and we answer by email. Migration help and custom work are sold separately.

💬 Open a Support Ticket

Try Live Demo