Sitefinity Web Forms widgets deprecation and migration paths

Overview and implications of the deprecation

Deprecated features

As of **release 14.1**, the **Web Forms widgets for front-end development** in Sitefinity DX are being deprecated. The following feature set will be in **sunset mode**:

- Built-in Web Forms widgets
- Web Forms widget templates
- Hybrid page templates
- Responsive design module

Recommended alternatives

Recommended alternatives are:

- .NET Core widgets and templates (official in release 14.1)
- MVC widgets and templates

License implications

How this deprecation affects your projects:

- Customers running legacy editions such as Standard, Marketing, Professional, Enterprise Editions prior to Sitefinity DX 14.0 will be affected, if they decide to upgrade to Sitefinity 14.1.
- Customers running current Sitefinity DX 14.0 + Enterprise Package customers who utilized Web Forms in their projects may also be affected, if they decide to upgrade to Sitefinity 14.1.

Deprecation plan

We are confident that the modern alternatives to the features that are being deprecated provide a greater value to both developers and practitioners. In addition, deprecating technologies that have reached their end of life will enable the Progress team to focus on further enhancing the platform and ensuring utmost stability, security, and performance.

With **release 14.1**, we have agreed on the following **deprecation plan**:

- For **newly created** projects on version 14.1: the Web Forms widgets and widget templates, Responsive design module, Hybrid page templates will **not** be available.
  We **do not recommend** using a deprecated feature set as part of any new deployment. We will not provide any additions, enhancements or modifications to the existing functionality.
- For **upgraded** projects (existing projects on versions prior to 14.1): these capabilities will continue to work, so that the projects are not broken upon upgrade and customers have more time to plan and execute their migration. The capabilities will be in **sunset mode,** which means that Progress will **not** invest in feature development or bug fixes.
- The deprecated feature set will not be eligible for support, security or bug fixes.

FAQ

- **Which technology should I start using for my next project – MVC or .NET Core?**

  A: We recommend you to carefully evaluate .NET Core and its features since it has many superior capabilities compared to MVC. Then make an informative decision about which technology better matches your requirements.

- **Will MVC be deprecated soon and require migrating my projects?**

  A: No, it will **not** be deprecated soon. We are planning to keep supporting it since it provides a great set of features and wide adoption by our customer's base.

- **Do you have any plans to support migration of existing widgets?**
  A: Yes, we will provide guidelines for migration of widgets.
- **Is .NET Core the recommended way to go from now on?**

  A: .NET Core is coded in a way that follows the latest programming patterns and allows us to deliver features quite fast (for example, new widgets development). The .NET Core renderer is not coupled with an old technology stack and will boost your developer productivity, it also provides superior capabilities compared to MVC. On the other hand, MVC has accumulated a significant set of widgets and features over the years, and they are quite handy. Also, your team already has experience with it and no learning curve will be ahead of them compared to .NET Core path.

- **Will existing MVC and Web Forms projects/pages continue running into .NET Core, so my team can slowly transition to .NET Core?**

  A: Yes, it is perfectly OK to use your existing MVC and Web Forms pages with .NET Core. In this case, the .NET Core renderer will act like a proxy and will serve those MVC pages to the client so you can have MVC, Web Forms, and .NET Core pages into the same project without a hustle. This approach helps especially when trying to migrate existing projects to .NET Core.

- **Do you plan to open source the widgets**?
  A: Yes, we will provide open-source .NET Core widgets with release 14.1 - having the code for them would boost developer productivity.

Why .NET Core

.NET Core is an open source, cross-platform framework, developed both by Microsoft and its community. It is a complete reform of ASP.NET that combines MVC structure and Web API into a single framework. This is the natural continuation of the .NET framework and Sitefinity will support the latest version of this framework by providing a decoupled frontend renderer.
There are multiple value propositions with our .NET Core implementation:

- Fast development and quick time to market
- Easier upgrades
- Complete isolation between website and administration
- Separated frontend scalability
- IDE of choice
- Cross platform deployment and development
- .NET Core is an open-source framework
- .NET Core Sitefinity widgets are open source
- Real WYSIWYG page editor
- Multiple widgets coming out of the box to support the needs of marketers
- Superior layout capabilities

How to kick-start with Sitefinity .NET Core

The following is an on-demand training course. You will learn how to use Sitefinity from a developer standpoint. Get familiar with the New UI and .NET Core in depth, learn how to develop widgets, templates, and backend extensions:

Foundations of Sitefinity Development

Advantages of using .NET Core renderer

Development performance comparison

Using .NET core renderer boosts development productivity by a large amount since the Renderer app is separate from Sitefinity CMS and has a fast startup time. Not restarting the CMS has a huge advantage in development, you could see your changes rapidly. You can take advantage of the dotnet run watch command to enable dynamic code changes and you can even take advantage of the hot module reload starting with .NET 6.

Development comparison

The development model stays the same by using the concept of widget development and template development as it was with the MVC and Web Forms renderers. In the renderer case, we are leveraging view components and layout files. There are samples and documentation available that demonstrate how this is implemented.

Extensibility support

Using .NET Core renderer, you can create your own layout files/templates and view components (widgets) as a distributed library. Moreover, any custom widgets distributed in widget libraries (including the out-of-the-box widgets provided by Sitefinity) can be extended by three ways:

- Changing the model class (holding the logic of the widget) through the DI container
- Adding new views
- Extending the *Entity* class and adding new properties to the widget designer

API support

The Renderer app uses a selected number of APIs that are concerned with content management only. They are used through the C# RestSdk package (Progress.Sitefintiy.RestSdk). This limits the room for error and hacks when it comes to developing on the frontend since the frontend is concerned only with content. Additional API endpoints can be exposed from Sitefinity CMS, if needed.

Performance comparison

Zero compilation with .NET Core as compared to MVC. Since the .NET Core Renderer is a standalone application, we can leverage the out-of-the-box async support with .NET.

CI/CD comparison

The renderer is extremely easy to deploy since it is a small app holding no data and a small number of configurations. This gives you the following options:

- Deploy only the Renderer app when having changes to the frontend only. Extremely fast deployment which reduces the downtime of the app.
- Point your Renderer app to the staging environment where all the content is pulled from live. This gives the ability to view your latest code changes against the latest live data.

Documentation and samples support

- GitHub samples repo
- Product documentation

Widget mapping

.NET Core widgets

- *Section* widget
  This is a layout widget that you can use to create the layout elements of your page. It also has out-of-the-box styling features, such as margins, paddings, and backgrounds. *Learn more...*
- *Navigation widget*
  This is the widget that you use to create your website navigation. *Learn more...*
- *Content block* widget
  This is the widget that you use to create content. You can use it to display text, tables, images, or videos. *Learn more...*
- *Content list* widget
  This is an out-of-the-box .NET Core widget that you use to render Sitefinity CMS content on the frontend - a single widget that can work with both static and dynamic content at the same time. *Learn more...*
- *Image* widget
  This is the widget that you use to display images. *Learn more...*
- *Call to action* widget
  This is the widget that you use to display one or two *call to action* buttons on your page. You can have a button for the primary action and a button for a secondary action. *Learn more...*
- Custom widgets
  With the new paradigm, you can create custom widgets extremely fast; the widget designers are autogenerated. *Learn more...*
- Login widgets and Forms
  These capabilities will be available out of the box with release 14.1. *Learn more...*

MVC and Web Forms widgets

View the list of available types of built-in widgets in Sitefinity CMS:
List of MVC and Web Forms widgets

Migrate your pages to .NET Core

The .NET Core Renderer does not limit you to create and edit only .NET Core pages. Using the Renderer, you can simultaneously work with .NET Core, MVC, and Web Forms pages. This means that in each site, you can have a mixed collection of all the three supported rendering mechanisms. This way, you can migrate your current website page by page.

For more information, see Migrate your pages to .NET Core.

Additional resources

- Sitefinity DX 14.0 and .NET Core: Better Get Ready
- Sitefinity's life cycle policy guide – view supported and retired versions schedule