Upgrading Sitefinity CMS

Best Practices

WHITEPAPER

Upgrading Sitefinity CMS

Best Practices

The common notion about upgrade is that of a complex process taking lots of time and resources, with an arbitrarily happy ending. But is that really true? Should we postpone upgrading our Sitefinity websites until the last possible moment, or should we upgrade more often? As a matter of fact, how much control do we have over the process and its outcome? And can't upgrade just be…simpler?

Honestly, upgrade is not a simple thing. It's a complex process involving multiple operations, including replacing Sitefinity precompiled logic, modifying database schema, configuration files, assembly references and so on. But this doesn't necessarily mean that performing an upgrade should be a hard task. We are constantly trying make as many elements of the upgrade process happen automatically for you. Upgrade should be easy if you are well informed, have a diligent approach and plan well in advance. This whitepaper shares our recommendations and best practices based on observations, in-house practices, and multiple conversations with customers. Moreover, we'll be trying to provide more context into what the upgrade process actually does, so that you're not following the listed steps blindly but have a deeper understanding of the mechanics of a Sitefinity website upgrade.

Before we begin, let's spend a minute talking about the top three benefits of upgrade, and why **we advise upgrading your Sitefinity website regularly.**

- Stay up to date.
  Even if you're not planning on upgrading right now, a critical security update or a feature that perfectly solves your business need will come up sometime in the future. Following the regular update cycle means less hassle when upgrading. Skipping multiple versions and then having to go through all of them means executing more upgrade scripts, handing more API changes, more testing, and so on. Simply put, upgrading your website regularly just makes things better.

- Keep it secure
  How do you keep your smartphone, personal computer, or webserver secure? Without even going into the complex network configurations, IP whitelisting, firewalls, antivirus software, and so on, everyone's priority is installing the latest software updates. Why would your CMS be any different?

- Benefit from the new features
  We ship amazing new functionality with each product release. Take a look at the release notes to get an idea of all the great stuff that gets out at least three times a year with Sitefinity CMS.

Let's look at how you can get these benefits by performing a smooth Sitefinity upgrade.

I. Preparation for the Upgrade

1. Plan sufficient time for the upgrade. Depending on different factors, like your project size, level of customization, automation and so forth, you might need to spend some extra time on code refactoring, functional testing, and so on. Additionally, planning sufficient time for the upgrade ensures there is enough time to resolve any unexpected issues that occur. If no issues occur and the upgrade completes smoothly, you can use the remaining time for user acceptance testing.
2. Review the release notes of all major versions between your current version and the target version
   - Review the API breaking changes between your current Sitefinity version and the version you are targeting for the upgrade. Note that when you are upgrading all listed breaking changes between the versions apply. In other words, if your site is currently running on version 9.0 and you want to upgrade it to 11.0, all breaking changes introduced in the releases between 9.0 and 11.0 apply to you.Despite our efforts to minimize API breaking changes, sometimes refactoring requires us to do so. Look for usage of the affected APIs in your project and build a plan for modifying any affected code after the upgrade. API changes are among the most common reasons for post-upgrade issues on customized projects. Anticipating them and having a clear plan for addressing the changes will make your upgrade much smoother.
   - Review the Database changes between the versions While not required, it is good to be familiar with the database changes that will be in place after the upgrade. The Sitefinity upgrade log file *(UpgradeTrace.log, located in ~/App_Data/Sitefinity/Logs)* reflects all changes applied to the database during upgrade and their status.
3. Check whether the target Sitefinity version is compatible with the .NET version running on your machine, and install any new version if required.
4. Install the target Sitefinity version and create a new project with that version locally. Make sure it is working properly. This will help isolate whether any issues pertain to your project specifically, or your environment needs additional configuration to meet the new version requirements.

5. Make a test deployment of the empty project on the target staging/test environment. Ensure the empty project deploys smoothly. Try to resolve any deployment issues. If you require assistance, open a support ticket for help. Ensuring an empty project deploys successfully will give you the confidence to later deploy the actual upgraded project more easily and only deal with project-specific issues (if any).
6. Make backups of the project files and corresponding database(s). Sitefinity upgrade is a non-reversible process, thus having a backup from a point in time right before the upgrade is a must. It enables you to restore your website to a backed up version if anything unexpected happens during the upgrade process, and guarantees data loss prevention.
7. Restore a copy of your website project (and corresponding database(s)), that will be upgraded, locally.
8. Change all connection strings and make them point to a local copy of the database(s). It might be a good idea to prohibit access of the local machine to staging and production database servers. This will ensure an error is thrown in case it tries to access any of those resources. Once the upgrade executes, the Sitefinity database is modified irreversibly (downgrade is not supported), thus ensuring that your test project is not connected to the live database is a must.
9. Ensure the pre-upgraded project builds and runs successfully on the local machine.
10. Decide on an upgrade mechanism (Project Manager or NuGet packages). Note that if the project is already using NuGet packages the upgrade needs to be done with NuGet packages.
11. Stop the local site from IIS.
    - If upgrading with Sitefinity Project Manager, and you have the project open in Visual Studio as well, close Visual Studio. This prevents potential lock of the Sitefinity project file *(SitefinityWebApp.csproj)* and ensures that Sitefinity Project Manager can update the file if needed.
12. Empty the Sitefinity logs folder *(~/App_Data/Sitefinity/Logs)*. You might need to stop the site from IIS/ Visual Studio/Sitefinity Project Manager, if the running worker process has locked some of the log files and prevents you from moving/deleting the old logs.
    It's best to have all logs generated from the time of the upgrade. That way you can identify more easily any upgrade or post-upgrade logged errors. If using custom logging mechanism (for example ELMAH), either stop it for the upgrade procedure, or make sure the local application can write logs and that they are accessible.

II. Upgrade Phase

Upgrading a Sitefinity project is essentially a three-stage process:

1. Replacing the **.dll* files
2. Adjusting the *web.config* and project references
3. Executing the upgrade scripts.

Replacing the *.dll files

Upgrade always begins with a change of the .dll files in the bin folder, containing the compiled Sitefinity logic. Depending on the selected upgrade mechanism, replacing the **.dll* files can happen in the following manner:

- The Sitefinity Project Manager directly replaces the existing .dll files in the /bin folder with new ones. By default, Sitefinity references its assemblies from the project /bin folder. If you have modified your project to reference **.dll* files from locations other than the /bin folder, once you build the project, the Visual Studio build process will replace the newly copied **.dll* files with the referenced ones, thus overriding the upgraded assemblies. To prevent this, you must make sure your project references its **.dll* files from the /bin folder.

- The upgrade with NuGet updates the package referenced version in your project packages.config file and restores the NuGet packages that are located in the Sitefinity NuGet server (http://nuget.sitefinity.com/nuget). The restore process downloads the new package versions to your ~/packages folder, replacing the existing ones. Later the Visual Studio build process replaces the .dll files in the /bin folder of the Sitefinity project. Keep that in mind that when using NuGet packages to upgrade, Sitefinity is not aware of the existence of the ~/packages folder or any other folders containing .dll files. It runs with the files present in the project /bin folder, thus you must build the project when using NuGet.

Adjusting the web.config and Project References

In some cases (depending on whether the version you are upgrading to requires it), you must apply changes to the *web.config* file. These changes are necessary to adjust for any new HTTP modules registrations, authentication mechanism changes, and so on. In a nutshell, these are changes utilizing ASP.NET compatible functionality, which are introduced in the new Sitefinity version, and must be reflected in the *web.config*. They do not necessarily get introduced with each new version, but when present they must be applied, otherwise your project will not behave in the expected manner. To get informed about the necessary *web.*

config changes for each version, refer to the Upgrade instructions in the Sitefinity documentation (https://www.progress.com/documentation/sitefinity-cms/upgrade).

- The Sitefinity Project Manager prompts to apply these changes automatically. You can leverage this functionality or decide to apply the changes manually (for example if you have modified the default *web.config*, and want to preserve your custom settings).

- When upgrading to Sitefinity versions 11.0 and above, NuGet takes care of adjusting the *web.config* changes automatically. If upgrading to version prior to Sitefinity 11.0, the upgrade with NuGet packages does not modify the *web.config*. In case such changes are needed, you must apply them manually when upgrading with NuGet packages. To get informed about the necessary *web.config* changes for each version, refer to the Upgrade instructions in the Sitefinity documentation (https://www.progress.com/documentation/sitefinity-cms/upgrade).

- In some cases, a change to the project references is needed. These changes get reflected in the SitefinityWebApp.csproj file. Such changes are necessary, for example when the new Sitefinity version introduces a new assembly.

- The Sitefinity Project Manager prompts to apply these changes automatically. A prerequisite for this is to have your Sitefinity project file named *SitefintiyWebApp.csproj* (the default name). If you have renamed your project file, the Sitefinity Project Manager will not be able to find it and cannot apply the changes automatically. You can leverage this functionality or decide to apply the changes manually.

- When upgrading with NuGet packages to Sitefinity versions 11.0 and later, the Sitefinity NuGet packages execute scripts to automatically adjust the project references. Similar to the Sitefinity Project Manager, in order for the NuGet packages script to automatically update your project references, you must have your Sitefinity project file named *SitefintiyWebApp.csproj* (the default name). If you have renamed your project file, the Sitefinity NuGet packages scripts will not be able to find it and cannot apply the changes automatically. If upgrading to a version prior to Sitefinity 11.0, these changes need to be done manually before upgrading the NuGet packages.

Addressing any API Breaking Changes in Your Code

Although not it's not officially considered a stage of the upgrade process, addressing the API breaking changes is of equal importance. Now that you have replaced the Sitefinity **.dll* files and adjusted the *web.config* and project references, you are ready to run your upgraded website for the first time. When you run the website, it will be using the new Sitefinity compiled logic. If you have extended your Sitefinity website using the Sitefinity API, you must check whether any of the APIs you are using have been changed. You can do that by going to the API changes in Sitefinity CMS page and reviewing all changes that apply to the versions between your pre-upgrade Sitefinity version, and the version you are upgrading to.

It's important to address any API breaking changes, apply any necessary binding redirects, and so on, prior to running your upgraded project for the first time. When you run the project for the first time the Sitefinity upgrade scripts take care of upgrading the database schema and configurations. If any runtime errors occur, due to API changes, this might disrupt the upgrade process and leave the database in an incomplete state. If this happens, you must apply the necessary changes in your code, restore the database to the point prior the upgrade and start the project, so the upgrade scripts can execute successfully.

Executing the Upgrade Scripts

The final stage of the upgrade process is to make the database schema and configurations upgrade. When the Sitefinity version you are upgrading to introduces changes in the persistence model or different default configurations, it ships with specific upgrade scripts as part of the responsible system modules. When you have successfully completed the replacing of the **.dll* files and modifying the *web.config* and project references, you must build and run your site. At this point your site will be running with the new version of the compiled code. Once it runs for the first time with the upgraded assemblies, Sitefinity CMS checks the version of each module in the system and detects (from the information stored in the database schema and configuration files) which modules are running on an older version and need to be upgraded. Each Sitefinity module that needs to be upgraded runs its upgrade scripts, modifies the database schema, and finally updates its entry in the database and configuration files to indicate that it's now running on the new version.

Summary of the Upgrade phase

In summary, here are the steps you must take to complete the upgrade process:

1. Make the "physical" upgrade on the local environment.
    - Using the Sitefinity Project manager, right-click on the desired project and select the Upgrade option form the context menu
    - By running *Update-Package Telerik.Sitefinity.All -Version [TargetVersion]* in the VS Package Manager console

2. Perform any additional steps, such as modifying the *web.config* file and/or project references, if needed.
3. Fix any code that needed changing due to breaking changes between the versions.
4. Build the project, verify it builds successfully.
5. Along the process, reference the upgrade troubleshooting section in the documentation: https://www.progress.com/documentation/sitefinity-cms/troubleshooting-upgrades
6. Run the project to make the upgrade scripts execute.
7. Apply the new license.

III. Confirmation and Testing Phase

This phase is important as it helps capturing any unexpected project behavior, runtime errors, and so on.

1. Make sure, the frontend of the site looks and feels the same, there are no obvious differences.
2. Make sure you can successfully log in to the backend.
3. Check the Administration -> Modules & Services page for any failed modules.
4. Check the Sitefinity logs. Pay special attention to the UpgradeTrace.log and check if there are any records indicating failed operation.

IV. Deployment Phase

1. Finally, you must decide on a deployment approach. You can either deploy the upgraded project files and restore the upgraded database on the target environment, or deploy just the project files and have the upgrade scripts execute and modify the live database automatically on the target environment.
    - Deploying the upgraded project files and restoring the upgraded database is the safer option, as you're deploying a copy of the setup you've already tested thoroughly. However, this option requires introducing content freeze from the moment you get the project locally and begin the upgrade process. Any changes after that will be lost. Alternatively, allow content updates for the first three phases, then freeze it, get another copy of the database, re-upgrade it locally and continue with the deployment. Additionally, any live content generation, taking place from the moment you back up the database for upgrade, to the moment you restore the upgraded database will be lost. Such changes may include user data input, such as forms, forum posts, comments, and so on.

    - Deploying just the project files will mean no content freeze is required as the upgrades scripts will execute against the target environment database. The downside is that the target environment will be down for a longer period and if any unexpected issues, caused by the difference in database content arise, they must be resolved fast, under greater pressure.
2. Plan for a deployment window. Ideally, deployment should be done during times of no or low(er) traffic to the site. This way you ensure ample time for the deployment and resolving any issues after that.
3. Perform the deployment.

Learn More