01

Technical Whitepaper

# Helix, Habitat and TDS Classic

## A How-To Guide

The Essential Collection
**Team Development for Sitecore**

Powered by Hedgehog

# Table of Contents

# Separation of Modules, as Promoted by Helix

Habitat implements Helix's 'separation of modules' by using separate projects in Visual Studio; this process attempts to avoid a scenario wherein developers break Helix principles by making it more difficult for them to create cross-module references. Enforcing this separation of modules could also be done by code-analysis tools. This is the reason why we've created FxCop Rules for Helix.

http://www.hhog.com/blog/sitecore-helix-fxcop-rules/

With these, it's possible for users to enforce correct module dependency direction by running these FxCop rules instead of relying on project separation. Additionally, users can, in theory, compact the projects but keep the modules separated by namespace or MVC Area and have the rules still enforce the modular separation from Helix principles.

Shrinking the number of projects is subtly mentioned in the Helix documentation, but doing so is widely approved by Sitecore's best practices team, including Thomas Eldblom, the creator of Helix. He has promoted FxCop rules in discussions around this topic.

When using module separation enforcement by project rather than by code analysis, developers can still break Helix principles by consciously making the reference (or accidentally make it through using ReSharper's auto-project-referencing feature). Using code analysis to enforce this is a much safer alternative, and shrinking the number of projects in a solution means that both build times and the site load times are improved.

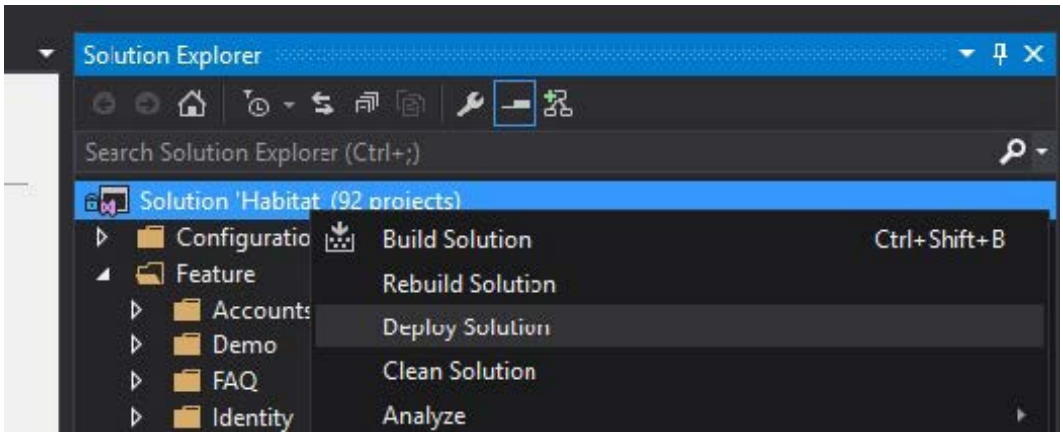
# Initial Base Repo Setup of TDS Classic Habitat

Hedgehog's TDS Classic copy of Habitat can be found on Github.

https://github.com/HedgehogDevelopment/Habitat/tree/TDS-latest

The simple procedure to deploy to a fresh Sitecore instance is as follows:

- **Set your settings**
- **Right click the solution, click Deploy Solution.**




This process both builds the code and deploys the items.

There is neither a requirement on NPM nor a need for running Gulp tasks; both are necessary in the original repo. Additionally, the initial build and deployment time is halved when using TDS Classic. A developer machine that clocked a 14-minute deployment time for the default Habitat solution clocked a 7-minute deployment time for the TDS Classic fork of the same repo.

## Updating Changes on your Local Site

When you receive the latest version from source control, which often contains other developers updated code and items, updating your local site is as easy as Right Click -> Deploy Solution.

If you want to push the items to Sitecore but not build/deploy the code, you can use the Quick Push feature (introduced in TDS Classic 5.6 and enhanced in TDS Classic 5.7) to work across the entire solution:

https://www.teamdevelopmentforsitecore.com/Blog/tds-5-6-quick-push

## Getting Updated Items from Sitecore

If, upon editing or adding multiple items across an entire Sitecore instance, you need to serialize them in order to add them into source control, you can add them using features in TDS Classic. Past versions of the product had features such as 'Get Sitecore Items' or 'Sync Sitecore' on individual projects.

TDS Classic 5.1 added a 'Sync Using History' View to the individual projects, which shows the most recently updated items in the sync window.

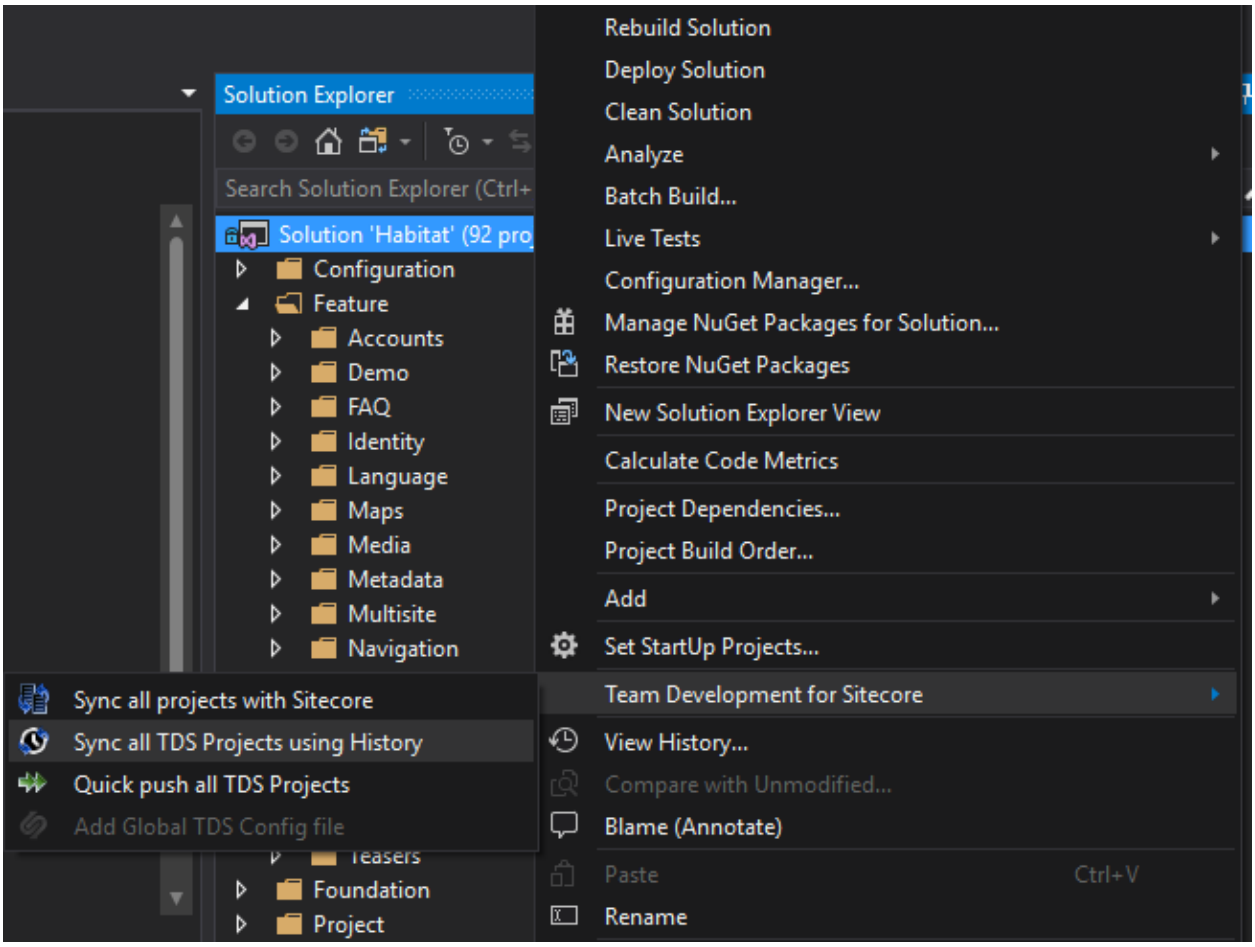
http://hedgehogdevelopment.github.io/tds/chapter4.html#sync-using-history-window

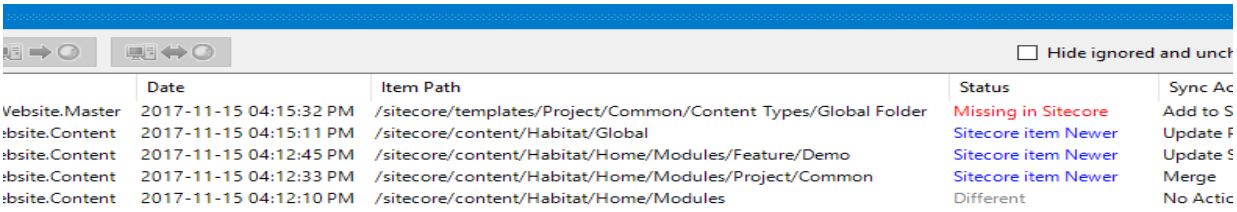
But with Habitat, users face the possibility that items are living across many TDS Classic projects in the solution (though this example is not limited to the Helix principles). TDS Classic 5.5 introduced a feature where you can 'Sync All TDS Projects using History' on the entire solution.

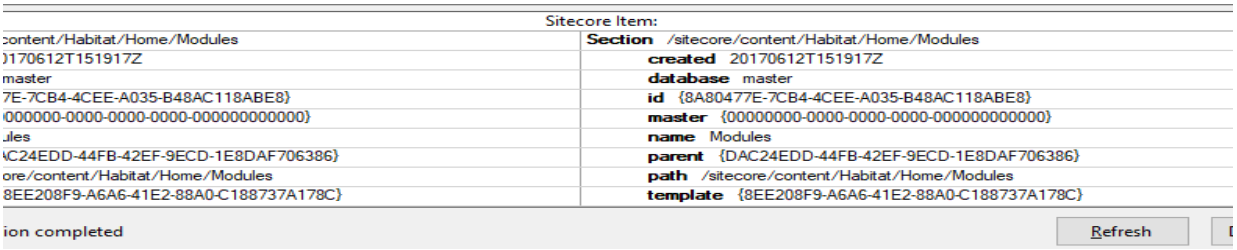
http://hedgehogdevelopment.github.io/tds/chapter4.html - sync-allprojects-using-history-window

This utilizes the same Sync Using History view as above, but covers all the TDS Classic projects in a solution.

## Sync Using History Windows



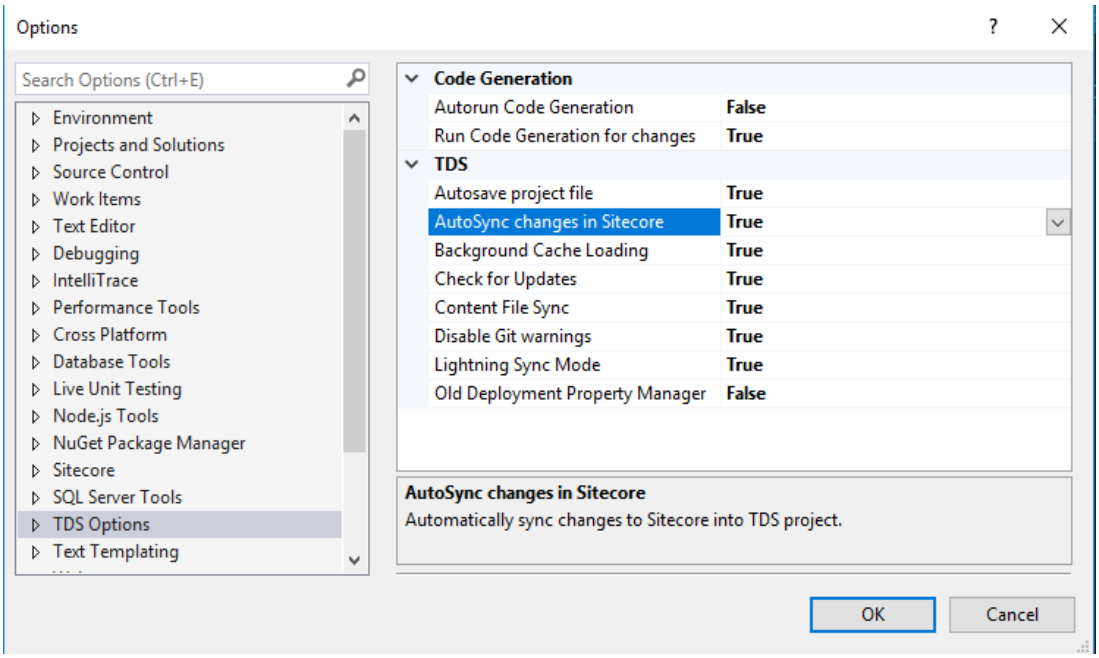
# Frequently Asked Questions

**"Is it possible to avoid a scenario in which I have to sync these items manually?"**

Yes. TDS Classic 5.5 introduced AutoSync which specifically helps with this.

Go to Tools/Options -> TDS Options/General, and turn 'AutoSync changes in Sitecore' to 'True'.

Now all item updates made in Sitecore will be automatically brought into your TDS Classic projects so long as your solution is open in Visual Studio.



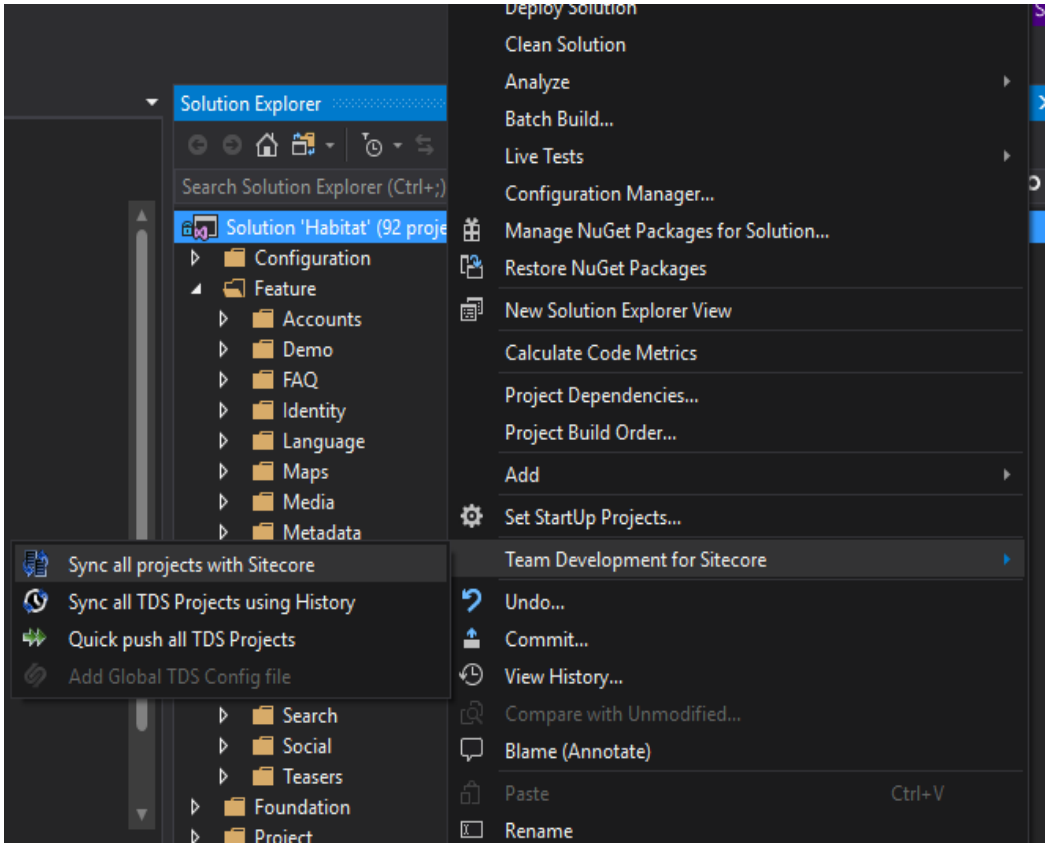

AutoSync from the Browser was new in TDS Classic 5.5; AutoSync itself has been around since TDS Classic 3.0.

http://www.seanholmesby.com/tds-the-evolution-of-auto-sync/

**"How can I view the differences between my file system and Sitecore across the entire solution?"**

Sync Using History and Auto Sync are great features for getting your own Sitecore changes onto disk. But if you grab another developer's changes from source control, you might want to compare disk items to Sitecore items.

TDS Classic 5.7 introduced the ability to Sync All Projects with Sitecore. You can now do a full comparison of every item across every project in your solution with a single click.



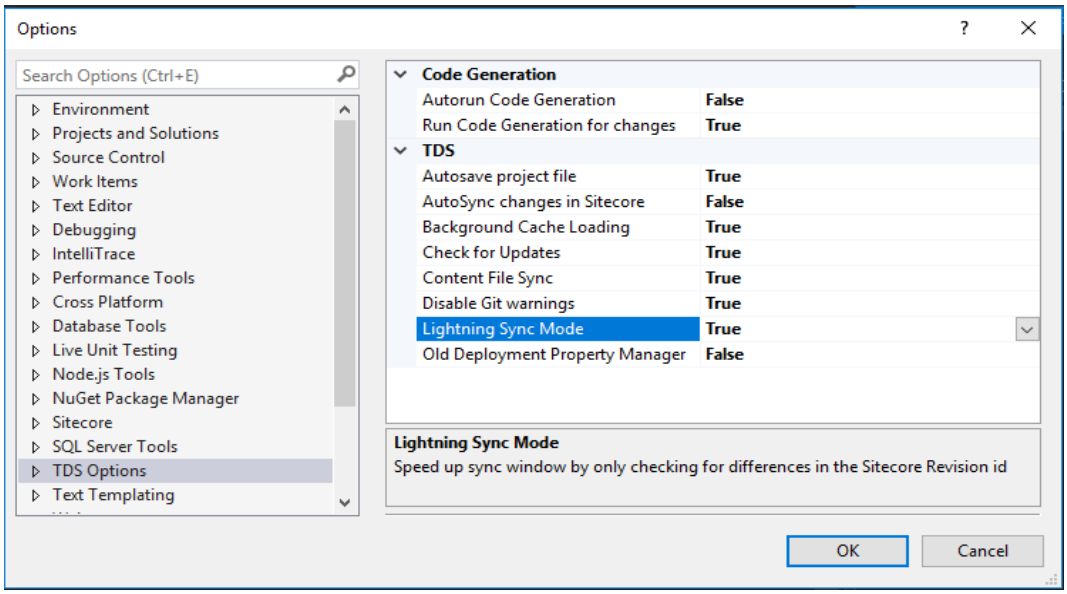

This utilizes the same Sync comparison window used in Sync with Sitecore, but it expands to cover every TDS project within the solution as well as all of the items changing within them.

**"Won't comparing so many items take a long time?"**

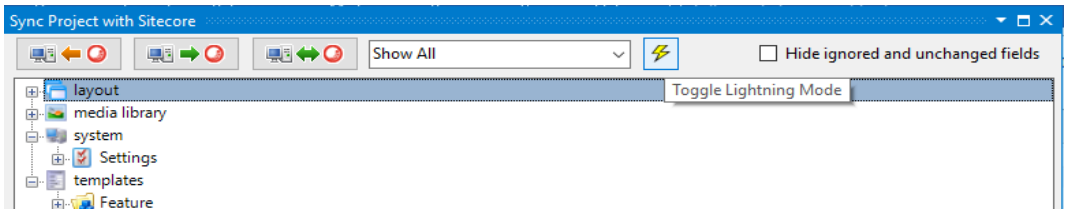
If you were doing a full solution sync with many items throughout all your TDS projects could take a long time, yes. We wanted to find a way to speed this up.

TDS Classic 5.7 introduces Lighting Sync Mode. Lightning Sync saves comparison time by initially comparing just the Revision ID of the item on disk to the Revision ID of the item within Sitecore. If the IDs match, the item is presumed to be unchanged and the sync window skips it without needing to compare each and every field on the item. This has led to full project syncs that are up to 300% faster than a default sync.

To enable Lightning Sync Mode, you can turn it on within the TDS Options Window:




Or by toggling on the Lightning Mode icon within the Sync window itself:



**"If Lightning Sync only compares Revision IDs, could it miss other item changes?"**

When editing content items, Sitecore will update the statistics on an item. That will include an update to the Revision ID, among other things such as Updated Date, Updated By, etc. Therefore, any normal content change will cause the Revision ID to change and Lightning Sync will pick it up.

However, it is possible that some changes could be overlooked in the comparison in the Revision ID is not updated. Luckily, this rarely happens and generally occurs only in edge-case scenarios.

A more detailed explanation of the edge cases for Lightning Sync can be found here:

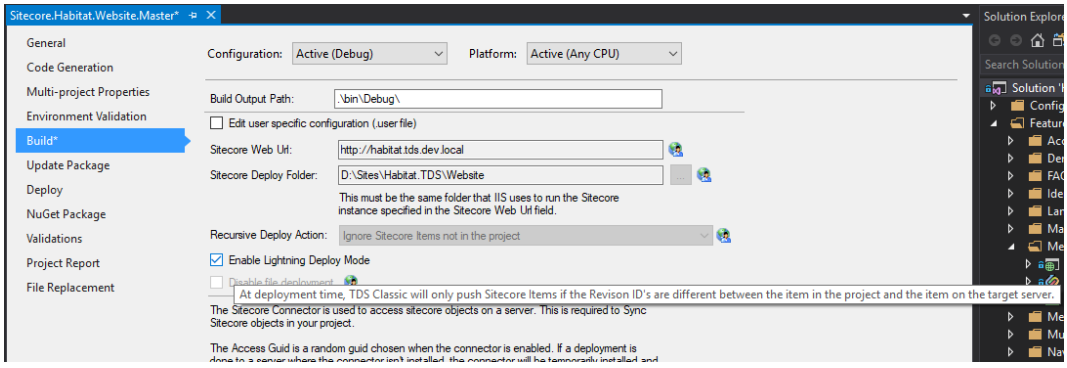
https://www.teamdevelopmentforsitecore.com/Blog/TDS-Classic-5-7-Lightning-Mode

**"Lightning Mode sounds amazing! Can it be used with any other features of TDS Classic?"**

Absolutely!

The Lightning Sync Mode setting found in the TDS Options above will apply to any hierarchical sync operation, including 'Sync all projects with Sitecore', 'Sync with Sitecore', 'Sync this item'. In addition to this, 'Quick Push all TDS Projects' and 'Quick Push Items' also adhere to the setting. If the Revision IDs match, Quick Push will skip over it, saving time by not 're-pushing' the item to Sitecore without any changes.

Local Deployments (using 'Deploy Solution') can also benefit from Lightning Mode as well, however this has a different setting, as it's an MSBuild operation, not a Visual Studio operation. To turn on Lightning Deploy Mode for local deployments, select the 'Enable Lightning Deploy Mode' checkbox on the Build Properties page, or by enabling it in the TdsGlobal.config file with <LightningDeployMode>True</LightningDeployMode>.

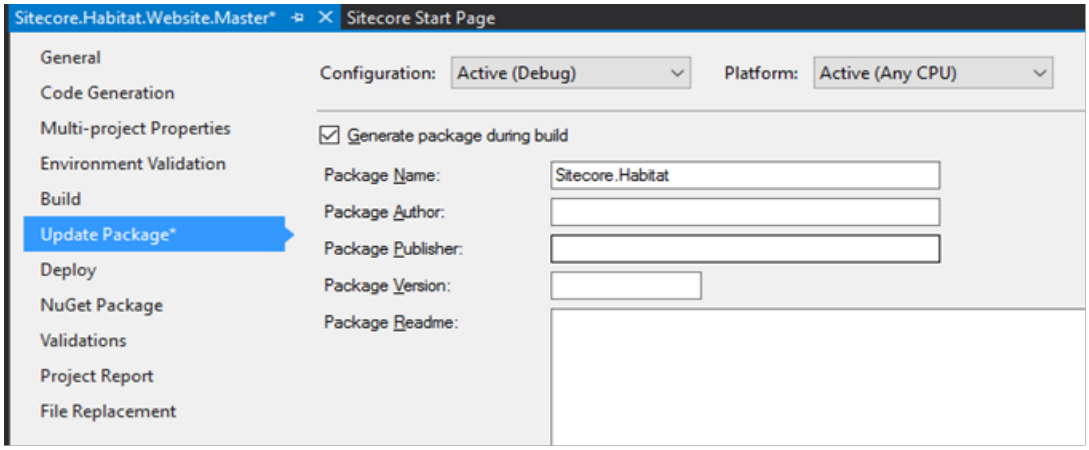



Lightning Deploy Mode is available for any configuration that uses the Sitecore Connector to deploy items, as it needs to access Sitecore to retrieve the Revision ID for comparison.

**"I want a single, deployable package for my builds. Is that possible with so many projects?"**

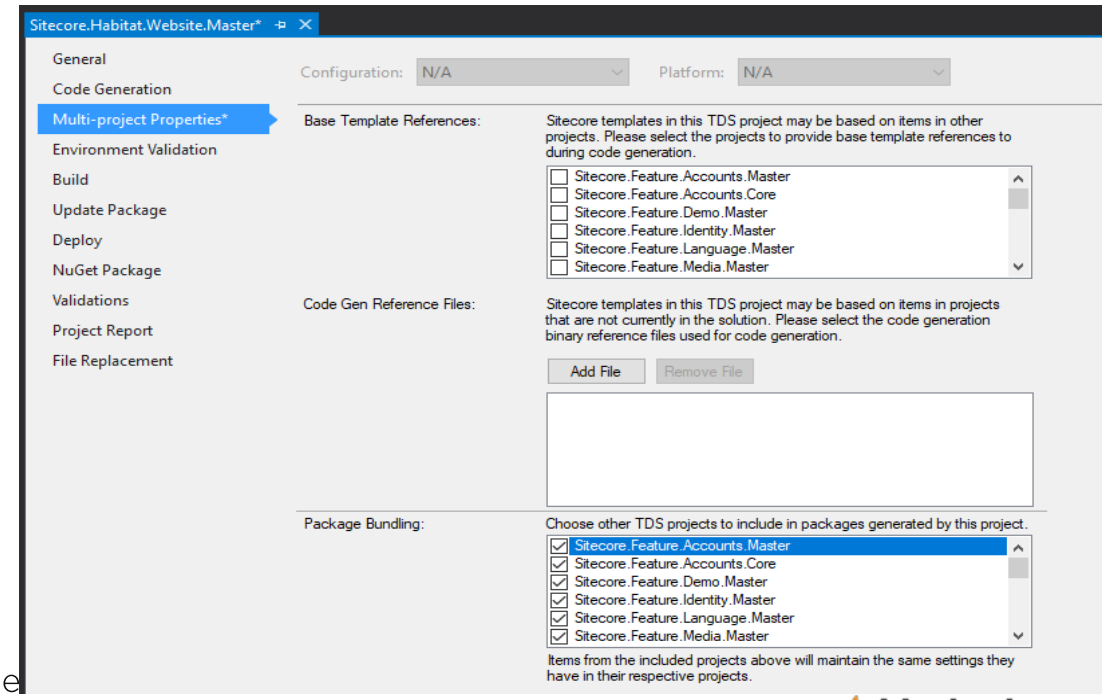
Yes. Creation of an Update Package on build is done by checking a checkbox on the project properties page. This feature was introduced in TDS Classic 2.0.

On the Properties page, choose the 'Update Package' tab, and turn on Package generation.




Since TDS Classic 5.0, the Multi-Project Properties page has featured a 'Package Bundling' section, where you can select the other TDS Classic projects you want to be included in the Update Package you're generating above.

Now you only need to manage one file for the entire solution's build, instead of a package for each project.



## Cross-Project Recommendation

This property page also has a section for 'Base Template References' where you can let your Code Generation (if you've turned it on) to be informed about the items contained in other projects.

In this way a user's code generation can be complete, with base classes/interfaces, that are generated from outside of the current project.

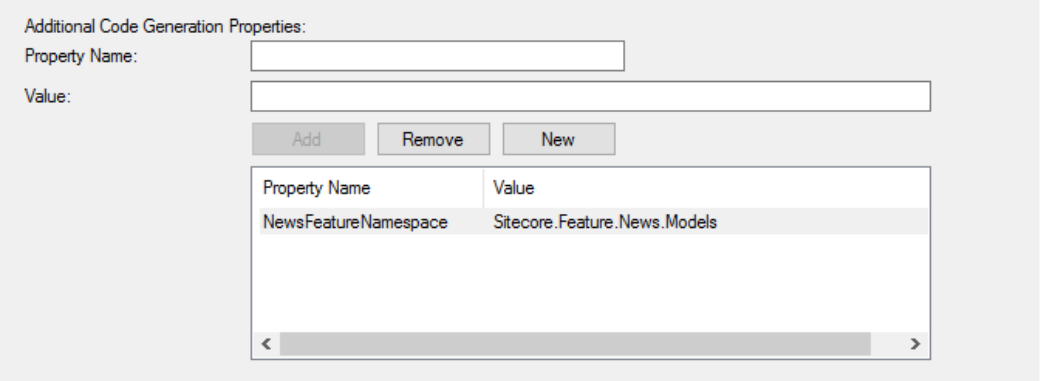
## Further Code-Generation, Cross-Project Specifics

Code generation in TDS Classic is very flexible, and users may generate code however they like. We have provided some examples of code gen on the hedgehog Github repo:

https://github.com/HedgehogDevelopment/tds-codegen

TDS Classic 5.6 introduced the ability for developers to add any additional custom properties to be used in their code generation.

In some cross-project code generation cases, a user may want to pass in custom namespaces that reference other projects. With TDS Classic 5.6, users can now pass their custom properties into the code generation, any fully customize the code-gen feature to your liking.



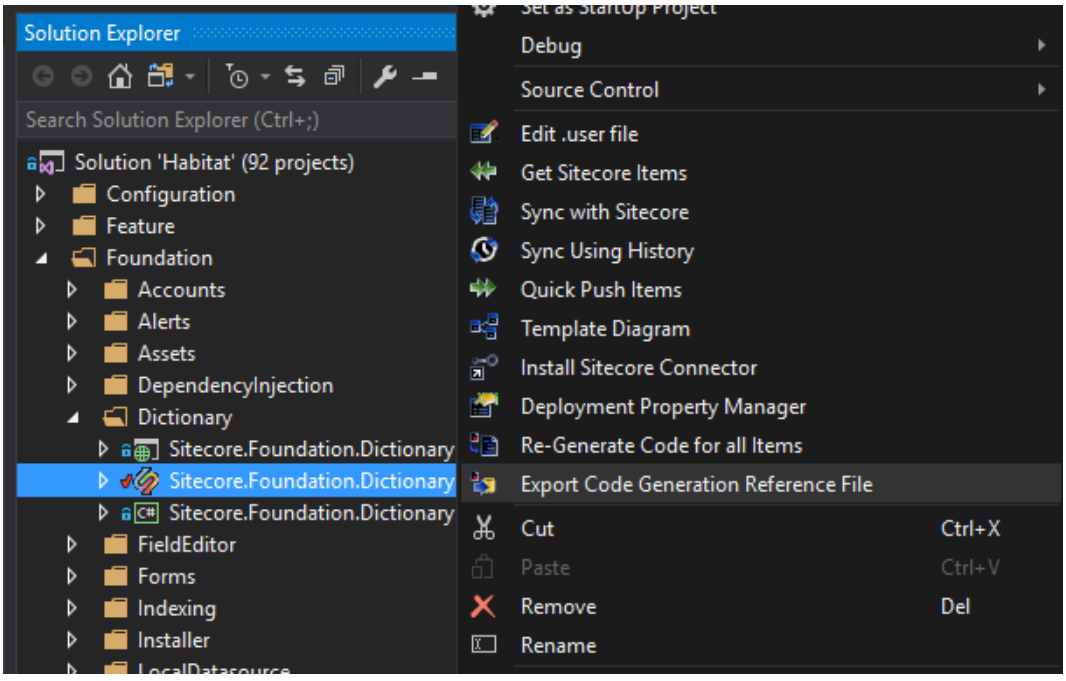

This is a more advanced feature of code generation, that can save the duplication of Header TT files for each project.

## Code-Generation From Another Solution

Using the Base Template References above, TDS projects can properly generate code with inheritance from other TDS projects, but this field only lists the TDS projects within the current solution.

Helix principles state that the code from a project can be split out into multiple, logical solutions. Therefore, users may have multiple solutions deploying to a Sitecore instance, each with their own TDS Projects. It's then possible that TDS projects in a 'Core' solution can have base templates and code that is needed in another solution.

TDS Classic 5.7 introduced the ability to 'Export Code Generation Reference File' from a TDS project that uses Code Generation, allowing it to be easily used in other projects to know about the base templates for inheritance.




That file can then be used in the inheriting solution using the TDS Properties – specifically by utilizing the 'Code Gen Reference Files' field on the 'Multi-Project Properties' tab.

## Benefits Summary

TDS Classic works fantastically with Helix-architected projects. It's used for serialization, but has many other features that greatly improve development efficiency. TDS Classic provides world-class, time and user tested features that can be enabled with a simple tick of a checkbox.

### Get A Live Demonstration

Email sales@hhog.com to schedule a live demonstration of the Sitecore Helix and overall performance improvements of TDS Classic 5.7.