This page contains best practices and tips for developing projects in Visual Studio in combination with a *Team Foundation Version Control* or *Git* source control repository and the Kentico continuous integration solution. Many of the recommendations also apply generally when using continuous integration with any type of source control system.

For detailed instructions and additional information, please refer to the following resources:

- Use Team Foundation Version Control
- Use Visual Studio with Git

Setting up workspaces

When using a *Team Foundation Version Control* repository, developers need to set up their Kentico project and *CIRepository* folder within a **Local** workspace. Server workspaces are not supported.

See the Decide between using a local or a server workspace article for more information about the workspace location types.

Creating and deleting objects

When you add a new object or delete an existing one in Kentico, the system creates or removes the corresponding XML files. You may need to perform additional actions to correctly track such changes, depending on the type of your source control repository:

- **Team Foundation Version Control** - by default, changes that create new files or delete existing ones appear in the list of *Excluded changes* in the *Visual Studio Team Explorer*. You need to view the detected *Add* or *Delete* operations and *Promote* the changes before you can check them in to the source control.

  Info

  Team Foundation Version Control by default excludes the **.class* and **.user* file extensions. These extensions appear as part of certain directory names in the *CIRepository* folder structure, for example, *cms.user* or *cms. class*. As a result, files contained in these directories will not show under *Pending* nor *Detected Changes* in the *Vi sual Studio Team Explorer*.

  To fix this issue, you need to add a .tfignore file with the **!*.*** rule to the root of the *CIRepository* folder, which tells the version control system not to ignore any changes made under the specified location.

- **Git** - when viewing *Changes,* files representing new objects appear in the *Untracked Files* section (delete operations are tracked automatically). You need to *Add* the files before you can *Commit* and *Push* them to a shared repository.

Changing object identifiers

Identifiers are values that the continuous integration solution uses to determine the names and locations of files representing objects in the *CIRepository* folder:

- In most cases, objects use their **code name** as the identifier.
- Object types that do not have a suitable code name use other values (a notable example are pages, which use their *alias path* as the primary identifier).
- Identifiers may also include values of related objects, such as the code name of the parent object, the name of the related site, etc.

Avoid making changes to object identifiers if possible, particularly for objects that have a large number of child objects or affect the identity of other objects (such as sites). This also includes operations that change the identifier indirectly, for example moving a child object under a different parent.

When an object's identifier changes, the continuous integration solution ***deletes*** the file representing the object in the file system and creates a new one based on the new identifier. This causes a ***loss of the file's version history*** in the source control and makes it difficult to merge changes made to the same object by other developers.

Checking in changes

When checking in (committing) changes to your source control, we strongly recommend including **all** detected file changes in the *CIRepository* folder. It may not be obvious which files correspond to specific changes made in the Kentico administration interface. Certain objects are represented by multiple files and changes may also affect related objects or relationships between objects.

To learn more about how objects are represented in the *CIRepository* folder, see the following pages:

- Continuous integration repository structure
- Object types supported by continuous integration

Getting new versions of the repository content

Whenever you load a different version of any files in the *CIRepository* folder from your source control, you need to:

1. Merge any files that conflict with your instance's local versions. See also: Resolving object conflicts
2. Restore the objects into your database.

The restore operation ensures that your instance's database is consistent with the file system repository. If you continue working on the instance without performing the restore operation, changes to local data may overwrite the new versions of the files that you loaded from the source control.

Resolving object conflicts

You may encounter object conflicts when getting or checking in *CIRepository* files. Conflicts happen if your local folder and the opposing folder contain a file with the same name, but different content. File names are based on unique identifiers of objects (in most cases the object's code name).

For more information about the identifiers of *CIRepository* files, see the Changing object identifiers section and Continuous integration repository structure page.

Planned conflicts can occur if you need to create ***the same object*** across your entire development environment. For example, multiple developers may create a web part category with the same name when starting development of custom web parts. Even if the objects have completely identical values for all configurable properties, a conflict will occur during synchronization, because the object's GUID (globally unique identifier) value is different for each developer.

The best solution is to have one developer create the object and check in the resulting XML files to the central source control. Other developers then load the XML files from the source control and restore the object to their local database. This approach completely avoids object conflicts and file merging.

If you cannot avoid an object conflict and need to merge XML files, always use the GUID value that is already checked in to the source control and replace your local GUID. Do NOT push your local GUID values to the source control – this could cause conflicts for all other developers in your environment.

Note:

- The XML elements and attributes used in the content of *CIRepository* files are ***case sensitive***. Make sure you preserve the letter case if you edit the XML content when merging files.

- If you change an object's GUID during conflict resolution and then restore the object to your database, you may need to manually fix broken references on your local development instance. For example, object and page fields that allow uploading of files store the GUID of the selected metafile or attachment object.

Unplanned conflicts occur if developers unintentionally create ***different objects*** of the same type with matching identifiers (typically code names) and then synchronize through the source control. In such cases, you need to contact the developer who checked in the conflicting XML file and one of you must delete and recreate the object with a different identifier (or rename the identifier values). We strongly recommend changing the local object that is not checked in to the source control yet, which prevents potential conflicts for other developers.

Undoing changes

We do not recommend using the Visual Studio **Undo** action for changes that create new files. Undoing changes may lead to inconsistencies between the state of the file system repository and the database. You can instead delete the given object in the Kentico interface, which also makes the corresponding changes in the file system.

If you undo a delete or modify operation and revert an XML file to its previous state, you then need to restore the objects into the database to reverse the object changes in Kentico.

Working with web application projects

For Kentico projects installed as a web application, the *App_Data/CIRepository* folder and its file content does not need to be included in the **CMSApp** project. Note that the folder still must be mapped to your source control (we strongly recommend having the entire Kentico project folder mapped).