When using the continuous integration solution, the system serializes database data into XML and stores the results on the file system in the project's **CMS\App_Data\CIRepository** folder.

The files are further organized within the following folder structure:

- **Site level folders** - the top level separates objects based on their relationships with sites: **@global** for global objects, **<site code name>** for individual sites (stores site-related objects, such as roles or ad-hoc page templates)
  - **Object type folders** - the next level contains folders for specific types of objects, based on ***object type names*** (see Object types supported by continuous integration to learn more).
    - **Object files** - individual XML files are named according to the ***code name*** of the corresponding object. Object types without code names use the value of a non-unique field, followed by a unique hash. Object types without any easily readable identifying fields use their GUID value for the file name.
    - **Binding data files** - binding data is not stored in separate files for individual binding records, but combined for each main object in the relationship. The binding XML data contains an identifier of the main object (parent) in the relationship, and then identifiers of all objects that have the given relationship with the main object, along with any other data stored by the binding. The binding files are named after the main object in format: ***<object name>@<unique hash>***
    - **Parent object folders** - object types that belong under a parent type (such as the transformations of a page type) use an additional level of subfolders – objects are separated into folders named according to the parent object in format: ***<parent object name>@<unique hash>***. For object types that have multiple possible parent types, the folder name also contains a ***<parent object type>_*** prefix (for example, transformations can belong under page types or custom tables).
    - **Separated field files** - certain types of objects store the values of specific data fields in separate files (placed next to the main XML file). For example, the binary data of metafile objects is stored in separate files. The separated files use names in format ***<object name>#<separated field identifier>*** and have an appropriate extension.

Long names and forbidden characters

Very long folder and file names automatically replace the middle of the name with **..** characters, preceded and followed by a limited number of characters from the start and end of the name. To ensure the uniqueness of the name, an **@** symbol followed by a fixed-length hash value is appended.

For example: *longcustomwpprefix..webpartcodenameend@fe6fd25d3e.xml*

If an object name contains characters that are not allowed in file names (for example slashes or backslashes), the given characters are removed and the same unique hash is appended.

Source control ignore rules

Certain source control systems may have ignore rules, such as *gitignore*, which exclude files or folders used in the continuous integration repository (for example the **.class* and **.user* file and folder extensions).

Such rules may prevent your environment from working correctly. We recommend that you evaluate ignore rules for the continuous integration repository location and disable them as required.

Example - CIRepository structure

Note: The example does not include all object types supported by continuous integration.

CMS\App_Data\CIRepository

- @global
    - cms.alternativeform
        - cms.news@8a478af656
            - filter.xml
    - cms.documenttype
        - cms.news.xml
    - cms.emailtemplate
        - blog.newcommentnotification.xml
    - cms.metafile
        - articles@7baacb97a4
            - articlesthumbnail.png@ead1667b00#file.png
            - articlesthumbnail.png@ead1667b00.xml
    - cms.pagetemplate
        - articles.xml
    - cms.pagetemplatesite
        - articles@7baacb97a4.xml
    - cms.transformation
        - cms.news@8a478af656
            - newslist.xml
    - cms.webpart
        - repeater.xml
        - logonform.xml
    - cms.webpartcategory
        - documentviewers.xml
    - cms.webpartlayout
        - logonform@9c9772813a
            - ecommercelogon.xml
- dancinggoat
    - cms.documenttypescope
        - articles@e0c9de07c4.xml
    - cms.emailtemplate
        - blog.newcommentnotification.xml
    - cms.pagetemplate
        - 3f0209c2-c0b9-47de-a855-0ca34e586b77.xml
    - cms.pagetemplatescope
        - dancinggoat-landingpage@5db220236b
            - home@6be8e032f6.xml
    - cms.role
        - contenteditor.xml
    - cms.userrole
        - contenteditor@f6aafc4899.xml

Storage structure for specific object types

Pages

The system uses three object types to store the definitions and settings of Pages. As a result, the files containing the serialized page data use a more complex structure than standard objects. The structure reflects how pages are stored in the database.

Each page is serialized in a site-specific **cms.document** folder and contains:

- A file in the *<nodealiaspath>@<hash>.xml* format describing the node data. This data determines where the page is stored in the content tree. The data is shared by all language versions of the page.
- A folder in the *<nodealiaspath>@<hash>#<culturecode>* format containing:

  - A *document.xml* file, containing language-specific data. Also contains the content of editable regions and editor widgets.
  - A *fields.xml* file, containing the data stored in the fields of individual page types.

Note: Other page-related object types, such as Page aliases, Page attachments and Page-level permissions (ACLs), are stored separately in the *CIRepository* folder.

For example, a *"Coffee Beverages Explained"* page with the "*en-us*" culture code on the *dancinggoat* site would be serialized into the following XML files:

CMS\App_Data\CIRepository

- dancinggoat
  - cms.attachment
    - articles_coffee-be..es-explained_en-us@b93096c0b8
      - coffee-beverages-explained-1080px.jpg@33fb4ffda2#file.jpg
      - coffee-beverages-explained-1080px.jpg@33fb4ffda2.xml
  - cms.document
    - articles_coffee-beverages-explained@e50cf58d9e.xml
    - articles_coffee-beverages-explained@e50cf58d9e#en-us
      - document.xml
      - fields.xml

Forms

The system uses multiple object types to store the definitions and settings of Forms. As a result, the files containing the serialized data of forms use a more complex structure than standard objects.

Each form is serialized into the following files:

- A global **cms.formclass** item.
- A site-specific **cms.form** item.
- Any alternative forms under the main form are serialized as child objects of the global *cms.formclass* item.
- Bindings between forms and roles (determine which user roles are allowed to manage the form and its data in the *Forms* application) are site-specific, and the main object in the relationship is the form's *cms.form* item.

Note: The continuous integration solution does NOT serialize the data records stored by individual forms.

For example, a form on the *dancinggoat* site with the *"UserFeedback"* code name, a *"VIPFeedback"* alternative form and role bindings would be serialized into the following XML files:

CMS\App_Data\CIRepository

- @global
  - cms.alternativeform
    - bizform.userfeedback@09642c63d0
      - vipfeedback.xml
  - cms.formclass
    - bizform.userfeedback.xml
- dancinggoat
  - cms.form
    - userfeedback.xml
  - cms.formrole
    - userfeedback@b8da61c184.xml

Users

Users in Kentico are a composite object type. As a result, the files containing the serialized data of users have a more complex structure than standard objects.

Each user is serialized into the following files:

- A global **cms.user** item, containing the basic user properties.
- A global **cms.usersettings** item organized as a child of the corresponding *cms.user* object. Contains the additional settings available when editing users in the *Users* application on the *Settings* tab.
- Other child objects and bindings, such as personal user categories and user-site relationships, are stored separately in the *CIRepository* folder.

For example, a user object with the username *Andy* would be serialized into the following basic XML files:

CMS\App_Data\CIRepository

- @global
    - cms.user
        - andy.xml
    - cms.usersettings
        - andy@7b3b12561e
            - 646cd67b-6380-462c-be47-d68449445465.xml

Resource strings

Each resource strings stored in the Kentico database (not strings in *resx* files) is serialized into the following files:

- A global **cms.resourcestring** item, containing the key of the resource string.
- A global **cms.resourcetranslation** item, containing bindings between the resource string and individual cultures. Stores the text value (translation) for each culture.

For example, a resource string with the *CustomString* key would be serialized into the following XML files:

CMS\App_Data\CIRepository

- @global
    - cms.resourcestring
        - customstring.xml
    - cms.resourcetranslation
        - customstring@09f9b53a99.xml