

The Kentico API provides a way to [serialize](#) many types of objects from the database into XML data. You can use the serialized data to synchronize objects between multiple development instances, in combination with a custom source control system or continuous integration solution. The system ensures that the XML data of matching objects is always identical and consistent (including element and attribute order), even when serialized on different instances of Kentico.

Serialization is supported for the object types listed on the [Object types supported by continuous integration](#) page.



Note: Only use the serialization API when implementing a custom synchronization solution. If you are using the default [Kentico continuous integration solution](#), you do not need to write any custom code.

Getting the serialized XML data for objects

To get the serialized XML data of Kentico objects:

1. Prepare a *BaseInfo* (*WebPartInfo*, *PageTemplateInfo* etc.) instance representing the object.
2. Call the **Serialize** extension method of the *BaseInfo* object (provided within the **CMS.DataEngine.Serialization** library).

The *Serialize* method returns an [XmlElement](#).

Example - Serializing a web part

```
using System.Xml;
using CMS.DataEngine.Serialization;
using CMS.PortalEngine;

...

// Gets a web part object
WebPartInfo webPartObject = WebPartInfoProvider.GetWebPartInfo("repeater");

// Creates an XmlElement containing the serialized data of the object
XmlElement objectElement = webPartObject.Serialize();

// Gets the serialized object data as an XmlDocument
XmlDocument xmlDoc = objectElement.OwnerDocument;
```

Deserializing XML data into objects

The API provides a way to deserialize XML data back into Kentico objects:

1. Read the XML data as an [XmlDocument](#) from the file system (or another source).
2. Get the [XmlElement](#) representing the given object from the *XmlDocument* (typically the root element).
3. Call the **Deserialize** extension method of the *XmlElement* (provided within the **CMS.DataEngine.Serialization** library).

The *Deserialize* method returns a **DeserializationResult** object, which you can convert to a specific *Info* object type (*WebPartInfo*, *PageTemplateInfo*, etc.) or a general *BaseInfo*. You can then work with the *Info* object in any way. For example, use the corresponding *InfoProvider* class to save the deserialized object into the application's database.

Example - Deserializing a web part

```
using System.Xml;
using CMS.DataEngine;
using CMS.DataEngine.Serialization;
using CMS.PortalEngine;

...

// Prepare an XmlDocument containing the serialized data of a web part
XmlDocument xmlDoc = ... ;

// Gets the root XmlElement of the XmlDocument
XmlElement xmlElement = xmlDoc.DocumentElement;

// Deserializes the XML data into a WebPartInfo object
DeserializationResult result = xmlElement.Deserialize();
WebPartInfo webPart = (WebPartInfo)result;

// Saves the deserialized web part into the database
WebPartInfoProvider.SetWebPartInfo(webPart);
```


IDs of deserialized objects

When deserializing XML data, the system automatically checks whether the given object already exists in the database (based on the code name or GUID values). If yes, the *Deserialize* method sets the appropriate object ID when creating the *DeserializationResult*. Saving the deserialized object into the database then updates the existing object instead of creating a new one.

Serializing all objects to the file system

You can call the API of the [Kentico continuous integration solution](#) to serialize all objects of the supported types directly to the file system.

Call the **StoreAll** static method of the **CMS.ContinuousIntegration.FileSystemRepositoryManager** class. The method returns a **RepositoryActionResult** object, which you can use to verify the success of the operation and process any errors that occurred.

 **Note:** The *StoreAll* method may have a long run time, depending on the number of objects in your database.

```
using CMS.ContinuousIntegration;

...

// Serializes all supported objects to the file system
RepositoryActionResult result = FileSystemRepositoryManager.StoreAll((logItem) =>
{
    // Optionally create a progress log by processing the 'logItem.Message'
    strings
}));

// Checks whether the serialization process was successful
if (!result.Success)
{
    foreach (string error in result.Errors)
    {
        // Display or log the errors
    }
}
```

The *StoreAll* method creates XML files containing the serialized data of objects in the project's **CMS\App_Data\CIRepository** folder. You need to ensure that the **CIRepository** folder is included in your source control system.

For details about the file system structure, see [Continuous integration repository structure](#).

Restoring objects from the file system

When developing websites in a team, you may often need to load the serialized XML data of objects from a source control system or another developer.

You can restore the data from the XML files in the project's **CIRepository** folder using the **ContinuousIntegration.exe** command line utility available in the Kentico project's **CMS\bin** folder. The utility deserializes the objects stored in the project's *CIRepository* folder and creates, overwrites or removes corresponding data in the given project's database. See [Restoring continuous integration files to the database](#) for details.



Warning: The restore operation *deletes* all objects of the supported types that do not exist as files in the repository. Only restore if you are sure that the *CIRepository* folder contains the required state of your object data. We also recommend creating regular backups of your database.

If you wish to implement a custom solution for restoring objects to the database, use the Kentico API.

Call the **RestoreAll** static method of the **CMS.ContinuousIntegration.FileSystemRepositoryManager** class. The method returns a **RepositoryActionResult** object, which you can use to verify the success of the operation and process any errors that occurred.



Notes

- To avoid potential collisions in the *CIRepository* folder, we strongly recommend calling the restore API from an external application while the main Kentico application is not running. For example, you can create a custom console application for this purpose (see [Using the Kentico API externally](#)).
- The *RestoreAll* method may have a long run time, depending on the number of objects in your file system repository.



```
using CMS.Base;
using CMS.ContinuousIntegration;

...

RepositoryActionResult result;

// Disables automatic processing of new smart search indexing tasks during the restore
operation to prevent potential conflicts
// Indexing tasks are still created, but processed after the restore is finished
using (new CMSActionContext { EnableSmartSearchIndexer = false })
{
    // Deserializes all supported objects from the file system repository and
    saves them to the database
    // Deletes objects from the database if their representation does not exist in
    the file system repository
    result = FileSystemRepositoryManager.RestoreAll((logItem) =>
    {
        // Optionally create a progress log by processing the 'logItem.
        Message' strings
    });
}

// Checks whether the restoring process was successful
if (!result.Success)
{
    foreach (string error in result.Errors)
    {
        // Display or log the errors
    }
}
```