Kentico CMS 7.0 Integration Guide

Table of Contents

Part

I

Introduction

1   Introduction

1.1   About this guide

This guide provides information about the Integration bus module in Kentico CMS. The main idea of the Integration bus module is to provide developers with the opportunity to integrate Kentico CMS with third party systems like CRMs or ERPs. The integration means synchronization of objects and documents in both directions. The data exchange is ensured by so called *connectors*. A connector is a common .NET class that needs to be implemented by a developer.
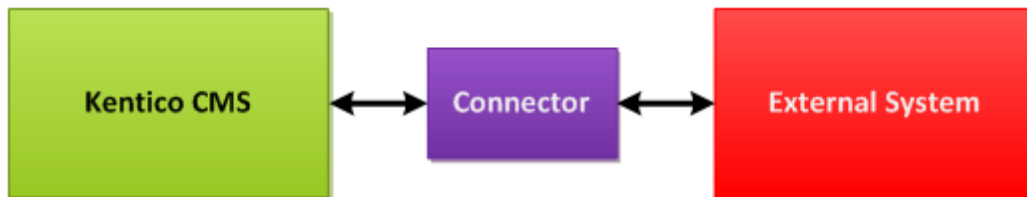
In the **Getting started** chapter of this guide, you can find basic information that helps you get started with the module:

- Related settings - explains settings related to the Integration bus module that can be adjusted in **Site Manger -> Settings -> Integration -> Integration bus**.
- Integration bus management UI - provides general information about the module's user interface in **Site Manager -> Administration -> Integration bus**.
- Enabling the sample integration connector - contains a step-by-step example explaining how to get the sample integration connector included in Kentico CMS functional.

In the **Concept** chapter, you can find general information about the concept of the Integration bus module and the principles on which it is based:

- Main idea - provides basic information about the module and about available modes of data transfer.
- Outgoing tasks (direction from Kentico CMS) - provides information about synchronization of changes from Kentico CMS to external systems.
- Incoming tasks (direction into Kentico CMS) - provides information about synchronization of changes from external systems to Kentico CMS.

In the **Connectors implementation** chapter, you can find in-depth information that you will need when implementing your integration connectors:

- Creating a connector class - describes how to prepare an operational skeleton of the connector class where the integration methods will be implemented.
- Implementation of outbound direction - describes implementation of methods ensuring integration in the outbound direction (from Kentico CMS).
- Implementation of inbound direction - described implementation of methods ensuring integration in the inbound direction (into Kentico CMS).

In the **Advanced scenarios** chapter, you can find the following single topic:

- Creating a custom subscription class - explains how a custom subscription class for synchronization of custom outgoing tasks can be created.

Finally, there are two reference topics at the end of the guide:

* Important types - provides information about data types that you will have to work with during your custom implementation.
* Database model - contains a database diagram and explanation of database tables used by the module.

Further reading

For information about how to quickly set up a simple integration connector, read Matt Swain's blog post about Kentico Integration Bus.

Part

II
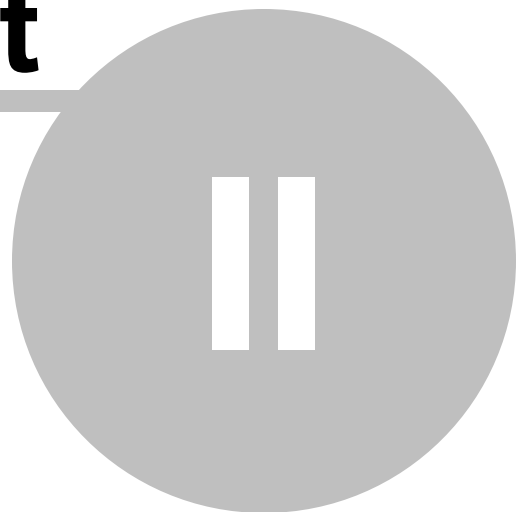
Getting started

2   Getting started

2.1   Related settings

Settings related to the System integration bus can be adjusted in **Site Manager -> Settings -> Integration -> Integration bus**. Here, the following settings can be adjusted:

Enable system integration bus	Indicates if the Integration bus logs and processes both incoming and outgoing integration tasks. Logging and processing of incoming and outgoing tasks can be enabled or disabled separately by means of the **Enable logging of incoming tasks**, **Enable processing of incoming tasks**, **Enable logging of outgoing tasks** and **Enable processing of outgoing tasks** settings below.
Enable logging of incoming tasks	Indicates if the Integration bus logs incoming integration tasks received from external systems. For this to work, the **Enable system integration bus** setting above must be enabled as well.
Enable processing on incoming tasks	Indicates if the Integration bus processes logged incoming integration tasks received from external systems. For this to work, the **Enable system integration bus** setting above must be enabled as well.
Enable logging of outgoing tasks	Indicates if the Integration bus logs document and object changes made in the Kentico CMS system as outgoing integration tasks. For this to work, the **Enable system integration bus** setting above must be enabled as well.
Enable processing of outgoing tasks	Indicates if the Integration bus processes outgoing tasks logged by document and object changes made in the Kentico CMS system. For this to work, the **Enable system integration bus** setting above must be enabled as well.

2.2   Integration bus management UI

User interface of the System integration bus module is located in **Site Manger -> Administration -> Integration bus**. In this part of the user interface, you can manage integration connectors and both outgoing and incoming integration tasks.

Integration connectors management UI

On the **Connectors** tab, you can see a list of defined integration connectors. An integration connector is a class which implements functionality for integration with a specific third party system.

New connectors can be defined after clicking 🌐 **New connector**. The following actions are available for each listed connector:

- **Edit** (✏️) - redirects you to an interface where properties of the connector can be edited.
- **Delete** (❌) - deletes the connector so that it is no longer defined and available.

Please note that when you add, edit or delete a connector, all defined connectors are reinitialized.
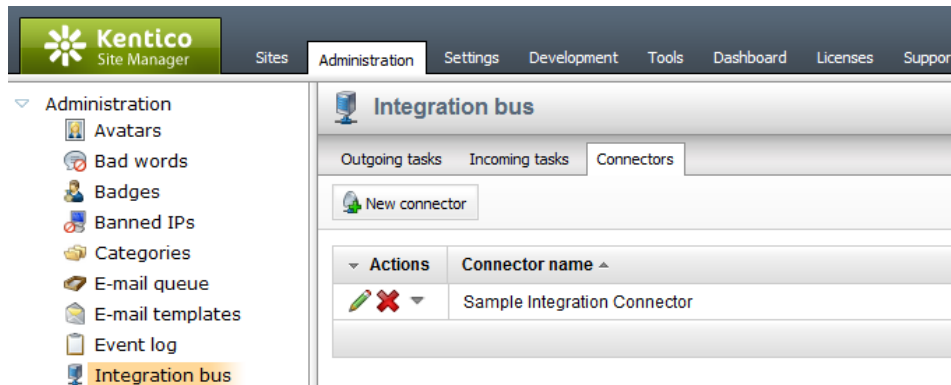

When creating a new integration connector or editing an existing one, the following properties of the connector need to be specified:

Display name	Name of the connector used in the system's administration interface.
Code name	Name of the connector used in website code. This name must match the value of the *ConnectorName* property declared in the connector's class. For more details, see step 1 and 4 in Getting started -> Enabling the sample integration connector.
Assembly name	Name of the assembly within the web project in which the connector's class resides. A connector can either be implemented in a standard assembly, or in *App_Code*. When it is stored in *App_Code*, you need to enter *App_Code* into this field and implement a module class with code that will ensure dynamic loading of the connector. For more details, see step 2 in Getting started -> Enabling the sample integration connector. Please note that if you installed the project as a web application, you should use *Old_App_Code* instead of *App_Code*.
Class name	Name of the class which implements the connector. If the class is located in a dedicated assembly (not in *App_Code*), the name should be entered including the namespace.
Enabled	Indicates if logging and processing of tasks by this connector is

| | enabled. Logging and processing of tasks needs to be enabled in **Site Manger -> Settings -> Integration -> Integration bus** as well in order for the connector to be functional. |
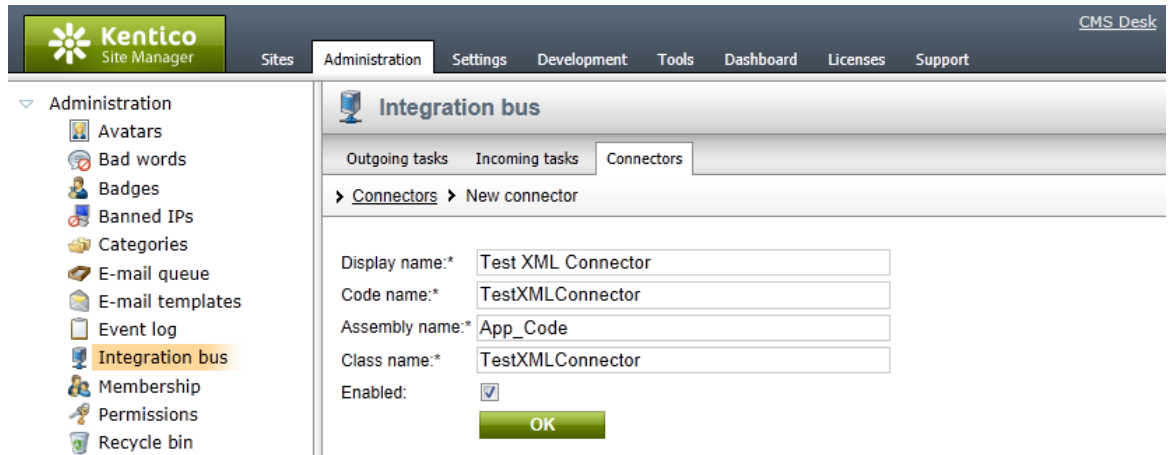
To make sure that a connector is loaded correctly, please see the listing of connectors. When something goes wrong you are notified by the warning (⚠) icon. When you move your mouse over the icon, you will be provided with a tooltip explaining the most common causes of failure.
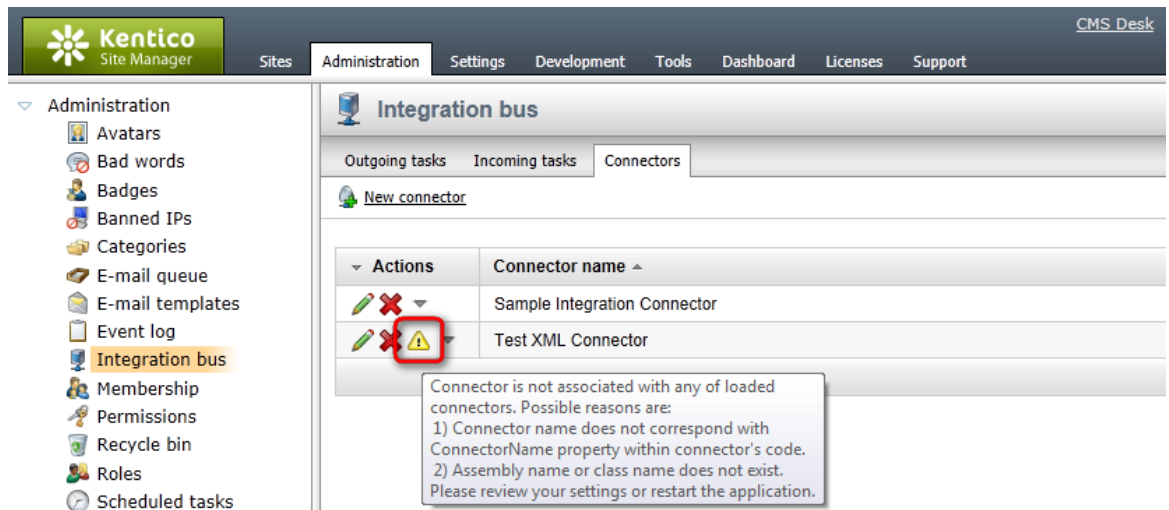
Outgoing tasks management UI

On the **Outgoing tasks** tab, you can see a list of logged outgoing synchronization tasks that were not processed yet. Outgoing tasks represent changes made in the Kentico CMS system that should be reflected in the target system with which Kentico CMS is integrated. By default, the tasks are listed from the oldest to the newest, i.e. in the order in which the will be processed.

Using the **Connector** drop-down list, you achieve that only tasks logged for the selected connector will be listed in the grid. When **(all)** is selected, all outgoing tasks logged for all connectors are displayed.

The following actions are available for each listed task:

- **View** (🔍) - displays object or document data transferred within the integration task in a new pop-up window.
- **Synchronize** (⚙) - starts processing of the respective single integration task.
- **Delete** (❌) - deletes the integration task without processing it.

In case that there are a large number of tasks listed, you can perform the above mentioned actions with more of them at once using the two drop-down lists below the grid. Using the first one, you need to chose whether to perform the action to all listed tasks, or only to the tasks selected by the check-boxes next to them. Then you need to choose the required action from the second drop-down list and click **OK** to perform it. If you perform synchronization of multiple tasks, they are performed in the order in which they were created, no matter how they are currently sorted in the grid.

Incoming tasks management UI

On the **Incoming tasks** tab, you can see a list of logged incoming synchronization tasks that were not processed yet. Incoming tasks represent changes in the external system with which Kentico CMS is integrated that should be reflected in the Kentico CMS system. By default, the tasks are listed from the oldest to the newest, i.e. in the order in which the will be processed.

Using the **Connector** drop-down list, you achieve that only tasks logged for the selected connector will be listed in the grid. When **(all)** is selected, all incoming tasks logged for all connectors are displayed.

The following actions are available for each task:

- **View** (🔍) - displays document or object data transferred within the integration task in a new pop-up window.
- **Synchronize** (⚙) - starts processing of the respective single integration task.
- **Delete** (❌) - deletes the integration task without processing it.

In case that there are a large number of tasks listed, you can perform the above mentioned actions with more of them at once using the two drop-down lists below the grid. Using the first one, you need to chose whether to perform the action to all listed tasks, or only to the tasks selected by the check-boxes next to them. Then you need to choose the required action from the second drop-down list and click **OK** to perform it. If you perform synchronization of multiple tasks, they are performed in the order in which they were created, no matter how they are currently sorted in the grid.

Integration task details

After clicking the **View** (🔍) icon on both of the tabs mentioned above, a pop-up window is displayed. In this window, you can see the object or document data transferred within the integration task.

Failed task processing details

When processing of an integration task fails, the **Failed** link is displayed in the **Result** column. If you mouse-over it, an error message explaining the issue is displayed. In case of outgoing tasks, it is the error message returned by the respective method implemented in the integration connector.

If you click the **Failed** link, a pop-up window is displayed, containing a **Synchronization log** with detailed information about individual attempts to process the synchronization task.

2.3   Enabling the sample integration connector

The default installation of Kentico CMS contains a sample integration connector. Two subscriptions for outgoing tasks are implemented in this connector — one for synchronization of all user objects, the second one for synchronization of all documents on all websites in the system. The connector's purpose is purely demonstrational — it only logs an event in **Site Manger -> Administration -> Event log** for each creation, modification or deletion of a user or document.

To add this sample to your web project:

1. Open your Kentico CMS installation directory (by default *C:\Program Files\KenticoCMS\<version>*).
2. Expand the *CodeSamples\App_Code Samples* sub-directory.
3. Copy the **Samples** folder into the **App_Code** folder of your web project.
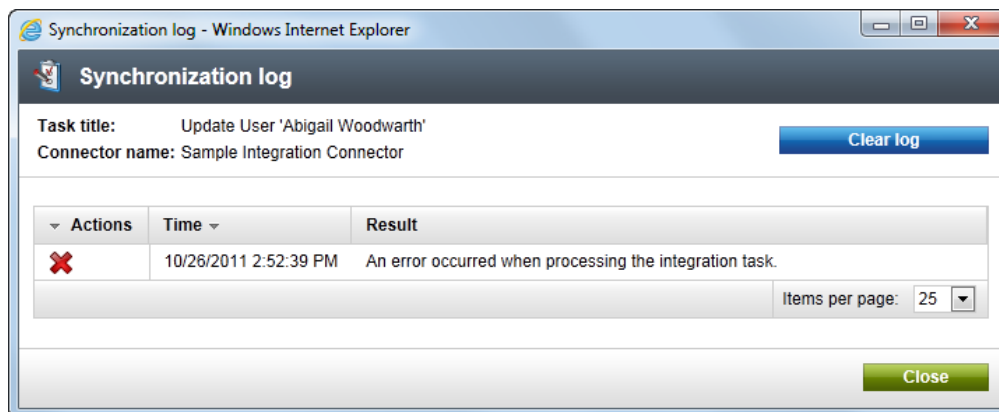

Web application installations

If your Kentico CMS project was installed in the web application format, copy the samples into the **Old_App_Code** folder instead.

You must also manually include the sample class files into the project:

1. Open your application in Visual Studio.
2. Click **Show all files** at the top of the Solution Explorer.

> 3. Expand the *Old_App_Code* folder, right-click the new **Samples** sub-folder and
> select **Include in Project**.

The following points summarize what needs to be done to enable the connector. Most of the steps have already been performed and are only described so that you can follow them when registering your own integration connectors.

1. Open your web project in Visual Studio, expand the *~/App_Code/Samples/Classes* folder and edit the *SampleIntegrationConnector.cs* class, which implements the sample connector. To make the connector functional, you need to ensure that the **ConnectorName** property initialized in the *Init()* method has the same value as the code name of the connector registered in the system. For the purposes of this example, change the value of the property to *SampleIntegrationConnector*.

```
public class SampleIntegrationConnector : BaseIntegrationConnector
{
    #region "Initialization (subscribing)"

    /// <summary>
    /// Initialize connector name and register subscriptions.
    /// </summary>
    public override void Init()
    {
        ConnectorName = "SampleIntegrationConnector";

        // Create subscription for all user objects
        ObjectIntegrationSubscription objSubscription = new
ObjectIntegrationSubscription(ConnectorName, TaskProcessTypeEnum.AsyncSnapshot,
TaskTypeEnum.All, null, PredefinedObjectType.USER, null);

        // Create subscription for all documents (on all sites)
        DocumentIntegrationSubscription docSubscription = new
DocumentIntegrationSubscription(ConnectorName,
TaskProcessTypeEnum.AsyncSimpleSnapshot, TaskTypeEnum.All, null, null, null,
null);

        // Register earlier created subscriptions for current connector
        SubscribeTo(objSubscription);
        SubscribeTo(docSubscription);

        // Please see implementation of ProcessInternalTaskAsync overloads (and
eventually comments for the rest of the methods)
    }

...
```

2. Expand the *~/App_Code/Samples/Modules* folder and view the code of the *SampleIntegrationModule.cs* class.

A connector class can either be implemented in a standard assembly, or in *App_Code* (like the sample connector). For connectors added to the App_Code folder, you need to ensure that the system loads the appropriate class when working with the connector. The *SampleIntegrationModule.cs* class demonstrates how you can do this. For additional information related to this topic, see Registering custom classes in App_Code in the Kentico CMS Developer's Guide.

```
using CMS.SettingsProvider;

[SampleIntegrationConnectorLoader]
public partial class CMSModuleLoader
{
    public class SampleIntegrationConnectorLoaderAttribute : CMSLoaderAttribute
    {
        /// <summary>
        /// Called automatically when the application starts.
        /// </summary>
        public override void Init()
        {
            // Assigns a handler for the OnGetCustomClass event.
            ClassHelper.OnGetCustomClass += GetCustomClass;
        }

        /// <summary>
        /// Gets a custom class object based on the given class name.
        /// </summary>
        private static void GetCustomClass(object sender, ClassEventArgs e)
        {
            if (e.Object == null)
            {
                // Checks the name of the requested class.
                switch (e.ClassName)
                {
                    // Gets an instance of the SampleIntegrationConnector class.
                    case "SampleIntegrationConnector":
                        e.Object = new SampleIntegrationConnector();
                        break;
                }
            }
        }
    }
}
```

In the case of integration connectors, the value of the **ClassName** property of the
ClassHelper_OnGetCustomClass handler's *ClassEventArgs* parameter matches the **Class name**
specified for the given connector object in the system.

3. Open the Kentico CMS interface, go to **Site Manager -> Administration -> Integration bus** and
select the **Connectors** tab. Click 🌏 **New connector**.
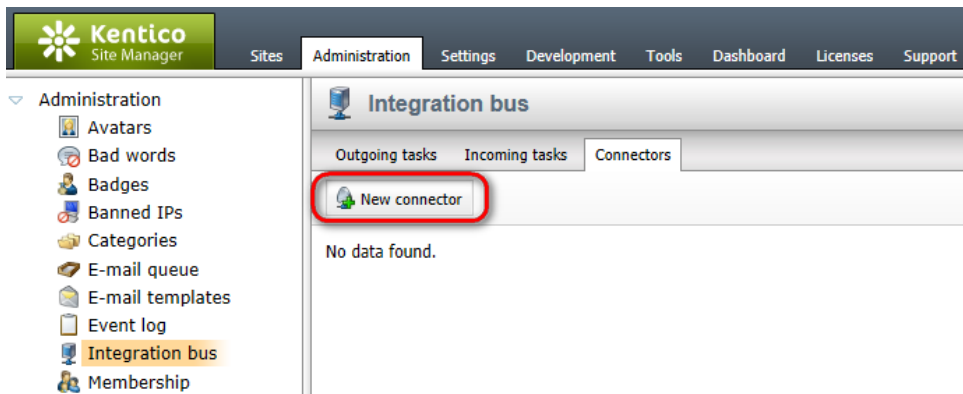
4. Fill in the following properties:

Display name	Name of the connector used in the system's administration interface.
Code name	Sets a unique name that serves as an identifier for the connector. This name must match the value of the **ConnectorName** property declared in the connector class (see step 1 above), so type in *SampleIntegrationConnector*.
Assembly name	Specifies the name of the assembly where the connector class is implemented. Enter *App_Code* for connectors implemented in the App_Code folder.
Class name	Specifies the exact class (including any namespaces) that defines the functionality of the connector. Enter *SampleIntegrationConnector* to load the sample connector class.
Enabled	Indicates if logging and processing of tasks by this connector is enabled. Logging and processing of tasks also needs to be enabled in **Site Manger -> Settings -> Integration -> Integration bus** in order for the connector to be functional.
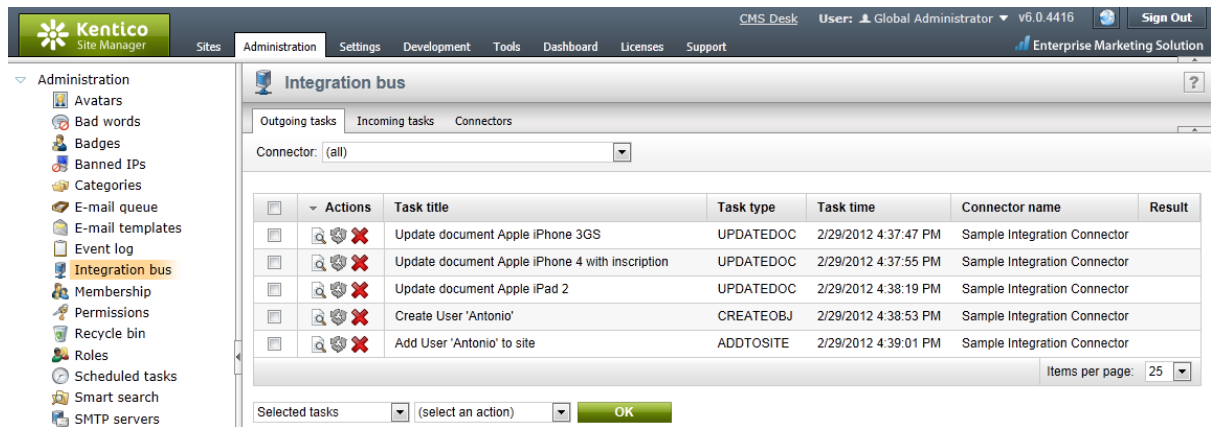
Click 🖫 **Save**.

5. Go to **Site Manger -> Settings -> Integration -> Integration bus** and adjust the settings of the module as follows:

- **Enable system integration bus**: enabled
- **Enable logging of incoming tasks**: disabled
- **Enable processing on incoming tasks**: disabled
- **Enable logging of outgoing tasks**: enabled
- **Enable processing of outgoing tasks**: disabled

The sample connector only handles outgoing tasks, so all incoming task settings may stay disabled. The reason why you should also leave processing of outgoing tasks disabled is only demonstrational — it will allow you to see the logged tasks in the UI in the following step. If you enabled them, the tasks would be processed right off and you would only see the logged events in the event log.

6. With the settings adjusted, try creating and modifying some documents and users. After doing so, go to **Site Manager -> Administration -> Integration bus**. You should see tasks for the respective actions logged on the **Outgoing tasks** tab. As processing of the tasks is disabled by settings, the **Synchronize** (🌐) action is grayed out and can not be performed at the moment.

7. Go back to **Site Manger -> Settings -> Integration -> Integration bus** and enable the **Enable processing of outgoing tasks** setting. Then, go back to **Site Manager -> Administration -> Integration bus -> Outgoing tasks**. The **Synchronize** (🌐) action should now be enabled. Instead of clicking the icon for each logged task, you can simply choose **All tasks** from the first drop-down list below the grid, choose **Synchronize** from the second one and click **OK**.

8. Once all tasks are performed, you can go to **Site Manger -> Administration -> Event log** and see the events logged by the tasks.

Part III

Concept

3 Concept

3.1 Main idea

The main idea of the Integration bus module is to provide developers with the opportunity to integrate Kentico CMS with third party systems like CRMs or ERPs. The integration means synchronization of objects and documents in both directions.
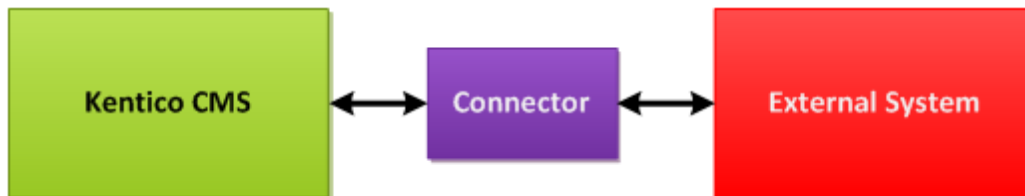
The data exchange is ensured by so called *connectors*. A connector is a common .NET class that needs to be implemented by a developer. In the [Connectors implementation](#) chapter, you can find more information on how to correctly implement such a connector class.

Data modes (data types)

The connectors are supposed to transfer some data. There are several supported data modes which can be understood as different amounts/volumes of transferred data. To proceed and understand the subsequent chapters, it is necessary to know the differences between these modes:

- **Simple** – use it when you are interested only in partial contents of an object (e.g. a text field), i.e. when you are not planning to synchronize the whole object.
- **SimpleSnapshot** – use this type when you are planning to synchronize whole objects and when you want to preserve foreign key bindings. This of course applies only when the 3rd party system has architecture and database design similar to Kentico CMS.
- **Snapshot** – this type is useful when you want to synchronize multiple objects at once. E.g. a main object with its children, e.g. polls together with poll answers.

3.2 Outgoing tasks (direction from Kentico CMS)

The outbound direction of integration is dependent on so called *subscriptions*. These allow you to specify which objects or documents you want to synchronize. When a user or the system itself performs an operation (such as create, update or delete) that corresponds with a subscription, the system passes the request to a connector either synchronously or asynchronously (this also depends on settings of the subscription).

Subscriptions

Subscriptions are used to define a scope over objects and documents. They can be basically understood as conditions – and when the conditions are met, the object or document is passed to further processing.

The following diagram illustrates how subscriptions fit into the whole integration process:
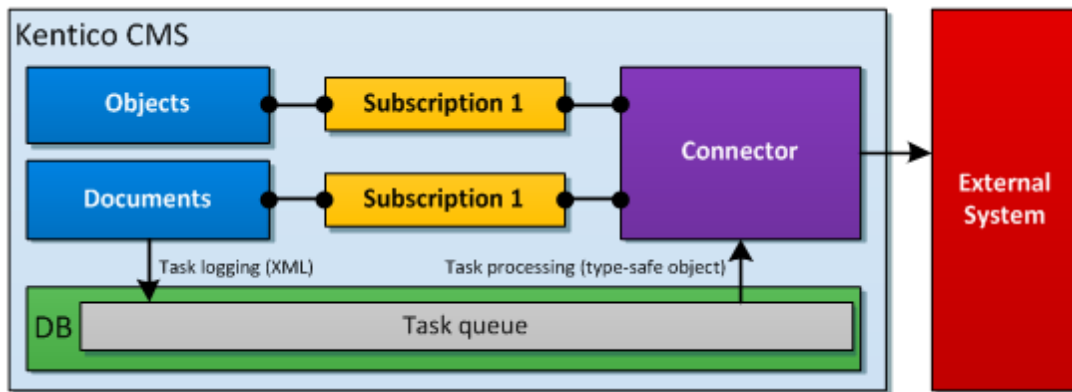
More details can be found in <u>Connectors -> Implementation of outbound direction</u>.

Types of processing

Unlike the inbound direction, the outbound direction supports two types of processing — <u>asynchronous</u> and <u>synchronous</u>.

Asynchronous processing

When you choose to process tasks asynchronously, object or document data are firstly stored in the database (we say that a *task was logged in the tasks queue*). Even if the external system is not currently accessible for some reason, the tasks are logged to a queue, which preserves the order of performed actions and prevents the synchronization from being lost. The tasks can be reliably processed once the external system is operative again. Another major advantage of this approach is that you don't lose the data of the tasks even if processing fails.
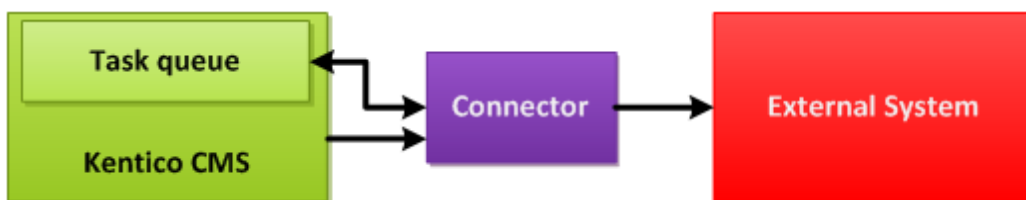
To ensure maximum performance, the logging and processing is postponed till the application reaches its *EndRequest* event, as can be seen in the following figure:
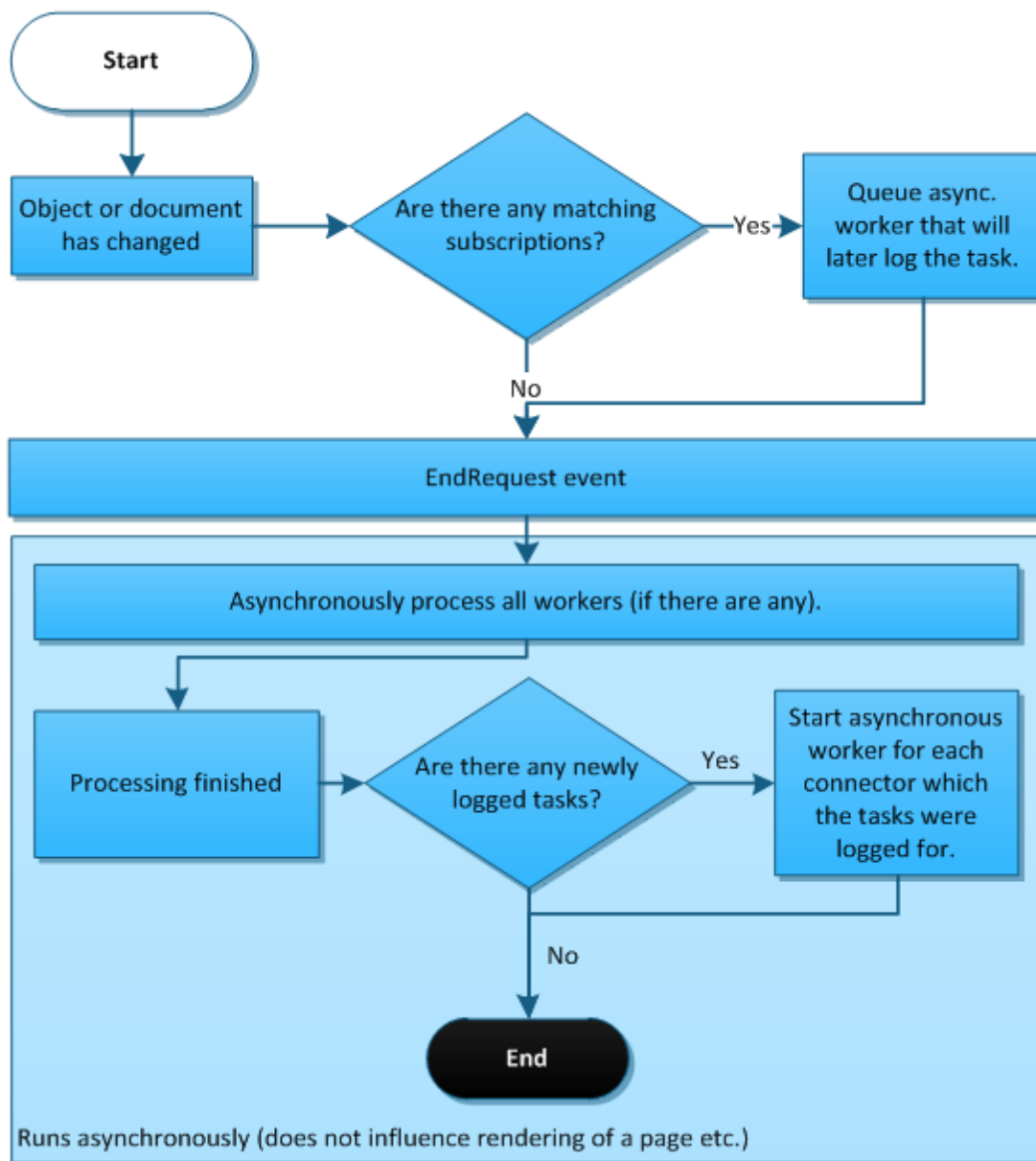
Asynchronous processing is highly scalable as all time-consuming operations are performed asynchronously. This approach doesn't have any major disadvantages and can be used for most scenarios.

As mentioned above, the processing starts basically on *EndRequest*. Alternatively, it can be launched by clicking **Synchronize** (🟢) in Site Manager -> Administration -> Integration bus -> Outgoing tasks. This can come in handy when the processing was previously turned off in settings or when the processing failed for some reason that has passed. Please note that processing doesn't start on *EndRequest* when the object or document has been changed in an asynchronous thread (e.g. in New site wizard when the **Log integration tasks** option is enabled). This limitation will be hopefully removed in one of the future versions of Kentico CMS.

When the processing thread starts, the connector starts to fetch tasks from the oldest to the newest (it is the classic queue principle). A fetched task is transformed to a strongly typed object and passed to

the methods implemented by the developer in the connector class. Some additional methods might be called, e.g. when a foreign key translation is desirable. When the task is processed, no matter whether successfully or not, the result value (of type <u>IntegrationProcessResultEnum</u>) is returned to notify the connector. Depending on the result, the connector decides what to do next.

The following scheme illustrates the whole procedure:
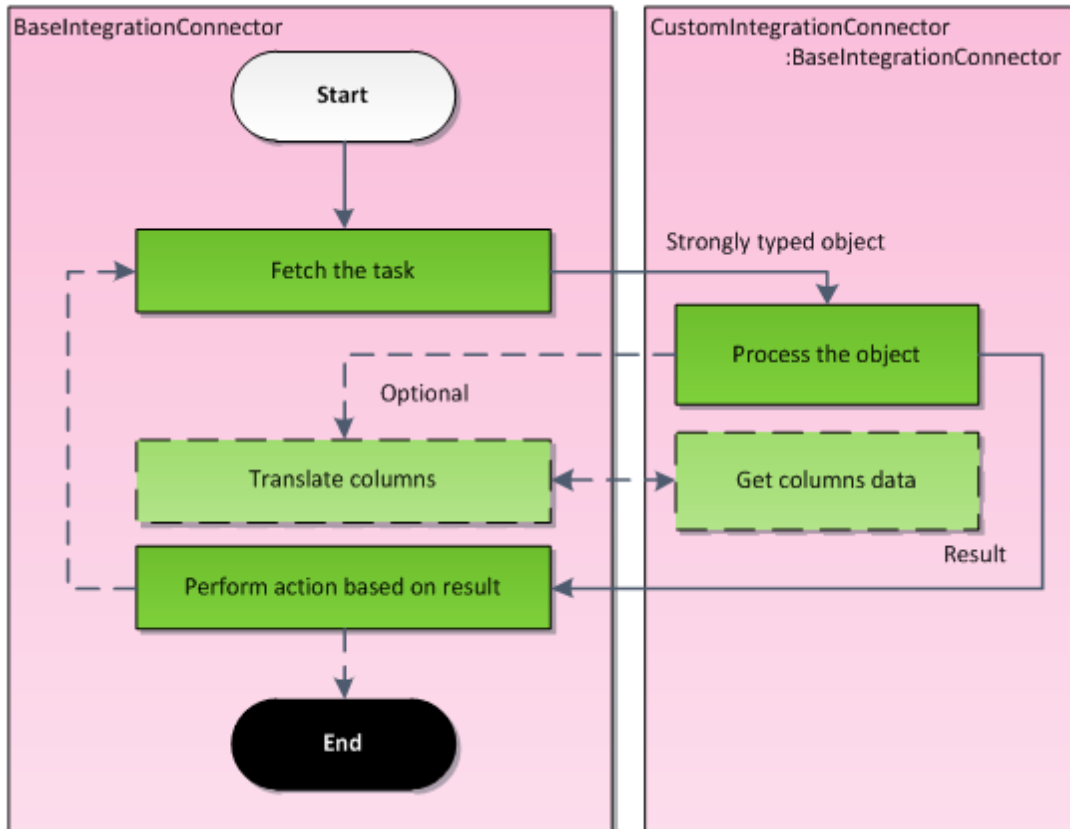
Please note that the *BaseIntegrationConnector* is already implemented. Your job is to prepare a code based upon this class (in the figure above, it is named *CustomIntegrationConnector*).

Synchronous processing

When using synchronous processing, the changed object goes directly to the connector for further processing:
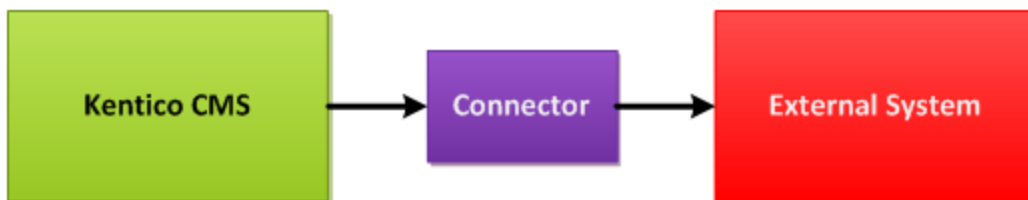
The following diagram illustrates the detailed order of the events:
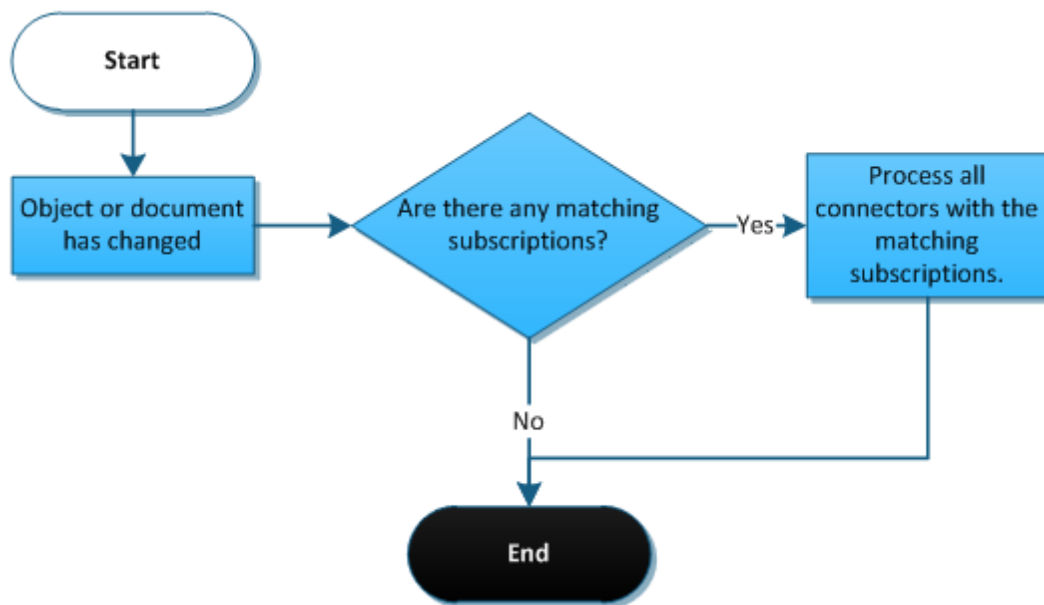
The major advantage of this attitude is that you can manipulate with the object in the context of Kentico CMS. This means that you can access properties like *Parent* or *Children* and the data are fetched from the database just in time. When it comes to documents, you can access properties like *Tags*, *Categories* and *Attachments*. On the other hand, you lose the option of persisting object and document data in the database (for the case when something goes wrong and the connector fails to process the request). Another disadvantage results from the nature of this type of synchronous processing — it slows down the page life cycle. It is recommended to use synchronous processing only when it is necessary for a specific reason.

Processing of synchronous tasks starts immediately after some object or document matching some subscription is changed. Unlike the asynchronous processing where the logging and processing is postponed till the application reaches *EndRequest*, this type of processing sends the data instantly to the subscribed connectors.
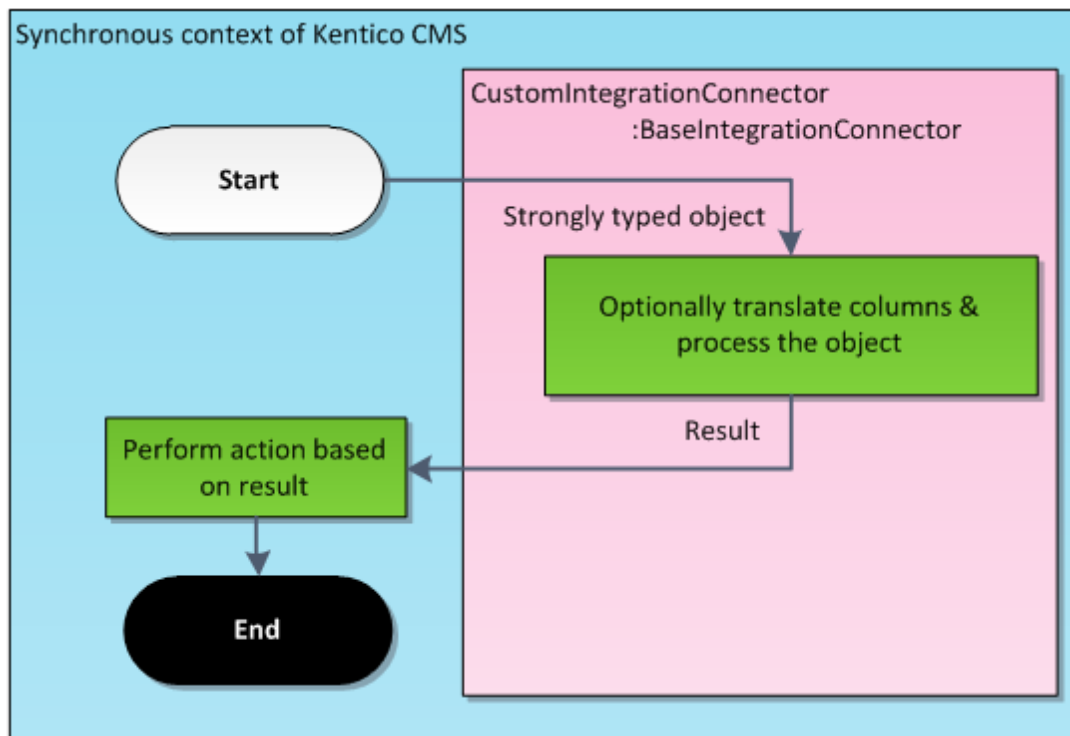
3.3 Incoming tasks (direction into Kentico CMS)

The inbound direction allows you to send data to the Integration bus and reflect them in Kentico CMS. The system stores the data in a queue, later takes it from the queue and processes it on a regular basis or on your request. This process is therefore always asynchronous.
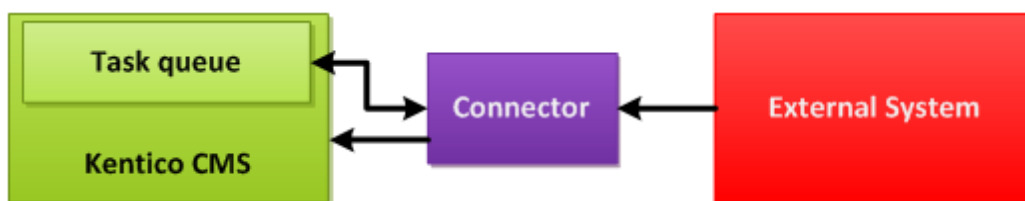
Generally, there are two approaches of implementing the inbound direction.

The first one assumes that the connector is placed within the 3rd party system and references DLLs of Kentico CMS. It also assumes that the Kentico database is accessible from the external system. The advantage is that even if the Kentico CMS instance is not accessible for some reason, the data of objects and documents (we call them tasks) are logged to the queue and can be reliably processed later without losing the synchronization.
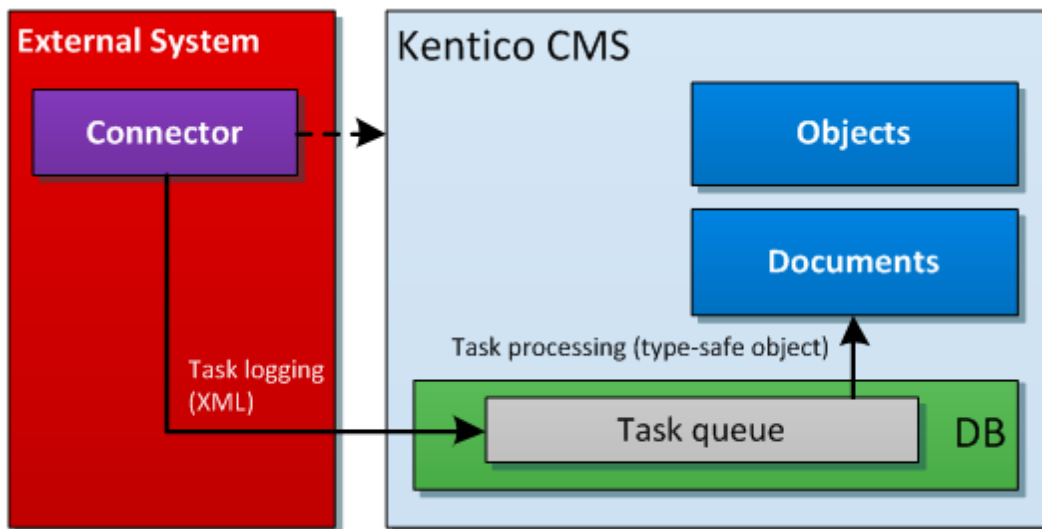
The second approach assumes that the connector is located within the Kentico CMS instance. The communication is ensured by a service (web or WCF). The advantage is that there is no need to reference Kentico CMS DLLs. On the other hand, you have to put an extra effort into implementation of the service.
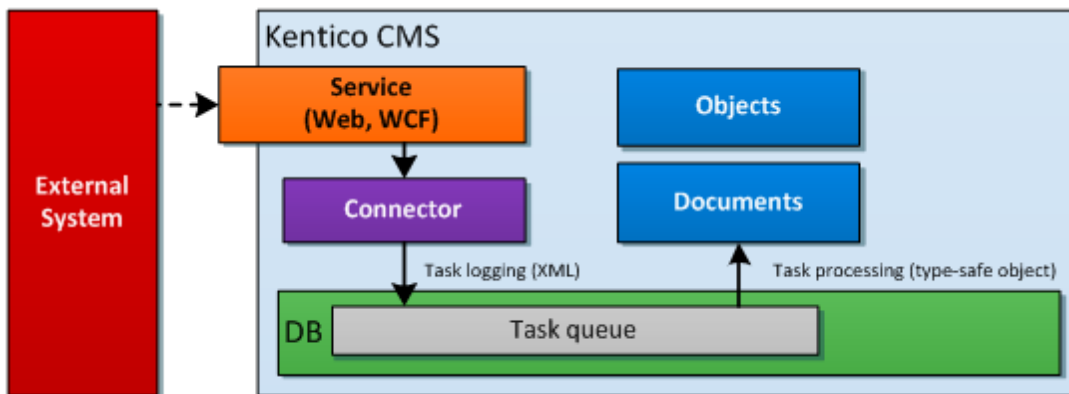
Whichever approach you choose, the processing of tasks is always asynchronous. It can be launched by:

- execution of the Process external integration tasks scheduled task.
- making a request to a special page.
- manually by clicking **Synchronize** (⟳) in Site Manager -> Administration -> Integration bus -> Incoming tasks.

Your job as a developer is to implement methods that will help the system log correct data to the queue. That means you have to convert the external object to a corresponding internal object or document and supply translation information if you want to preserve foreign key bindings:
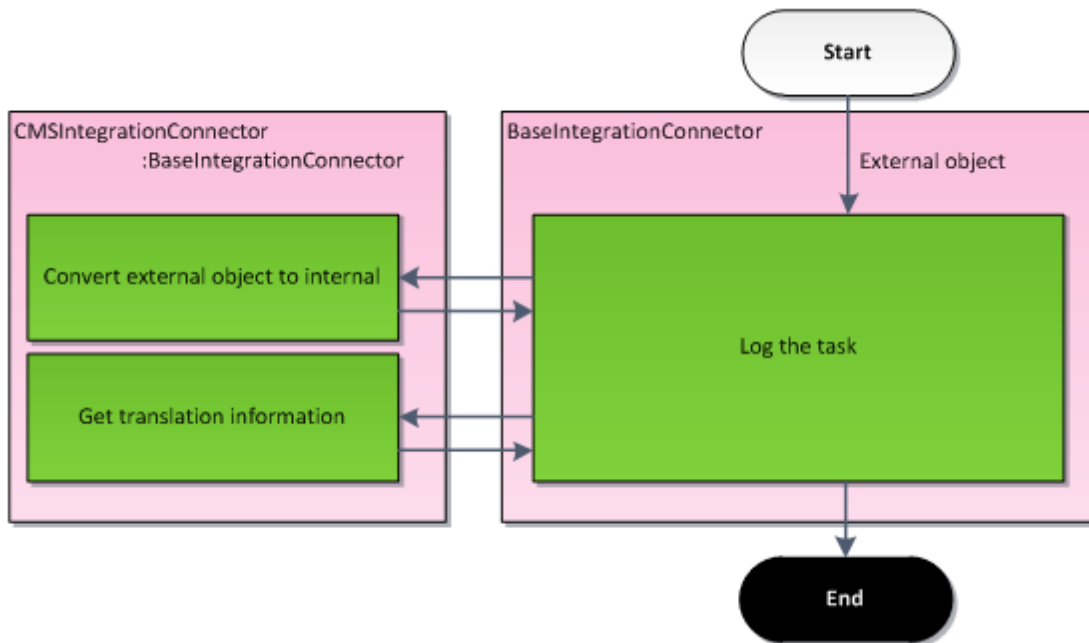
Scheduled task

In **Site manager -> Administration -> Scheduled tasks**, you can find a scheduled task called **Process external integration tasks**. By default, its execution is planned once a day. Please adjust the planning according to your needs, while it's recommended to perform the synchronization at least on an hourly basis.

You can also use the scheduled task for manual initiation of external tasks processing by clicking **Execute** (▶).

Part

IV

Connectors implementation

4   Connectors implementation

4.1   Creating a connector class

A connector can be located either in separate assembly or in *App_Code* (or *Old_App_Code* if you installed the project as a web application).

Creating a connector in App_Code

If you decide to create a connector in *App_Code* (or *Old_App_Code*), please see the instructions in Enabling the sample integration connector.

Creating a connector in a separate assembly

If you decide to create a connector in a separate assembly, you need to go through the following steps:

1. Open the Kentico CMS solution in Visual Studio and add new project to it (name it e.g. *CustomIntegrationConnector*). This will ensure the connector will have its own DLL assembly.

2. It's necessary to add references to following namespaces:

- CMSSynchronization
- SettingsProvider
- SynchronizationEngine
- DocumentEngine

You will also probably utilize the following namespaces:

- CMSHelper
- DataEngine
- GlobalHelper
- SiteProvider
- WorkflowEngine

3. Add one new class as shown in the figure below. This class has to inherit from *BaseIntegrationConnector*.
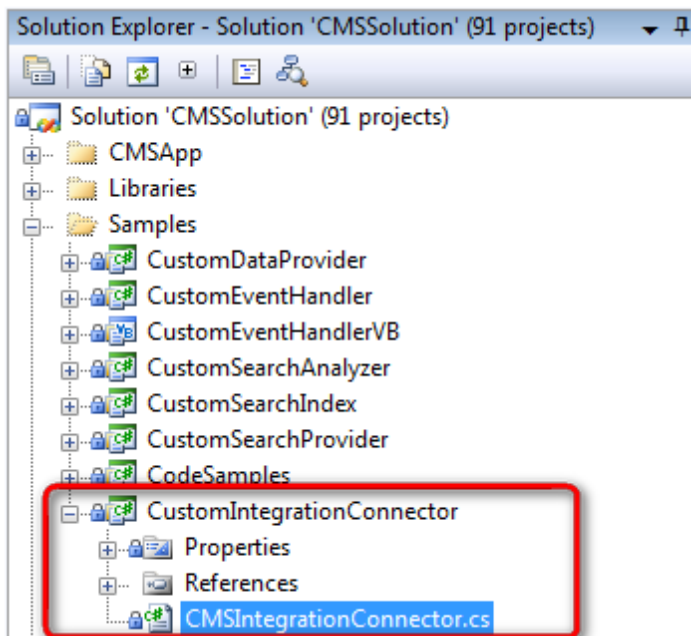
4. Override the *Init()* method and set the *ConnectorName* property within this method. The value of the *ConnectorName* property should be equal to the connector's code name defined in the user interface. At this point, it is also recommended to register the connector in the Kentico CMS system. Registration of an assembly with a connector class is explained in the Integration bus management UI topic.

5. Build the solution in Visual Studio.

6. Navigate to **Site Manager -> Integration bus -> Connectors** and verify that the connector has been loaded successfully (no warning icon (⚠) is displayed next to it).

4.2   Implementation of outbound direction

At this point you should be decided:

- which objects and documents you want to synchronize
- which data type you want to use
- whether you want to use the synchronous or the asynchronous mode

Based on this information, you will be able to decide which methods you need to implement. You should also have an operational skeleton of the connector prepared as described in the Creating a connector class topic.

Subscribing

After initializing the *ConnectorName* property, the second thing you have to do within the *Init()* method is to create subscriptions. Subscriptions are classes inheriting from *AbstractIntegrationSubscription*. The subscription determines the scope of tracked changes to be synchronized. By default, you can use any of the three prepared subscription classes – *BaseIntegrationSubscription*, *ObjectIntegrationSubscription* and *DocumentIntegrationSubscription*.

Later, you will see that it is not always necessary to create subscription objects. These only give you the highest level of scope specification granularity.

The particular subscription classes provide the following filtering options:

- **BaseIntegrationSubscription**
  - **SiteName** – string determining the name of he site which the object or document is bound to, e.g. *CorpSite*. You can also use *AbstractIntegrationSubscription.GLOBAL_OBJECTS* to subscribe only to global objects.
  - **TaskType** – enumeration determining the type of task (create, update, delete, etc.). See Important types -> Enumerations -> TaskTypeEnum for more details.

- **ObjectIntegrationSubscription**
  - **ObjectType** – string determining the type of the object (e.g. *cms.user*).
  - **ObjectCodeName** – string determining the code name of the object (eg. *administrator*).

- **DocumentIntegrationSubscription**
  - **DocumentNodeAliasPath** – string determining the path to the document (e.g. */Home*).
  - **DocumentCultureCode** – string determining the document's culture (e.g. *en-%*).
  - **DocumentClassName** – string determining the document's document type (e.g. *CMS.MenuItem*).

You could notice the usage of the percent character (%) above. It is a wildcard representing 0-n arbitrary characters. This wildcard can be used within any of the string parameters. In the culture code example above, it specifies all English cultures (*en-US*, *en-GB*, etc.).

If you don't want to constrain some string field, just leave it *null* in the constructor. If you don't want to constrain the task type, use the value *All*.

You can either build your own subscription objects and register them, or take advantage of built-in helper methods which will do the job for you.

Building your own subscription objects

You can use the following examples as an intuition or as a basis for building your own subscriptions.

A) Using the following example, you can create two subscriptions. objSub and docSub. The *objSub* subscription covers objects whose object type starts with *poll.poll* - both polls themselves and their answers. The docSub subscription covers all actions done with documents on *NewSite* located under / Home/ in the content tree.

```
ObjectIntegrationSubscription objSub = new ObjectIntegrationSubscription
(ConnectorName, TaskProcessTypeEnum.AsyncSnapshot, TaskTypeEnum.CreateObject,
"PersonalSite", "poll.poll%", null);

DocumentIntegrationSubscription docSub = new DocumentIntegrationSubscription
(ConnectorName, TaskProcessTypeEnum.AsyncSimpleSnapshot, TaskTypeEnum.All,
"NewSite", "/Home/%", null, null);

SubscribeTo(objSub);
SubscribeTo(docSub);
```

B) Using the following example, you can create two subscriptions—customTableSubscription and custTableSubscription. The customTableSubscription subscription covers changes made to a custom table where "customtable.SampleTable" refers to its code name. The custTableSubscription subscription covers the modifications in the data of a custom table where "customtable.SampleTable" refers to its code name.

```
ObjectIntegrationSubscription customTableSubscirption = new
ObjectIntegrationSubscription(ConnectorName,
TaskProcessTypeEnum.AsyncSimpleSnapshot, TaskTypeEnum.All, null, null,
"customtable.SampleTable");

ObjectIntegrationSubscription custTableSubscirption = new
ObjectIntegrationSubscription(ConnectorName,
TaskProcessTypeEnum.AsyncSimpleSnapshot, TaskTypeEnum.All, null,
CustomTableItemProvider.GetObjectType("customtable.SampleTable"), null);

SubscribeTo(custTableSubscirption);
SubscribeTo(customTableSubscirption);
```

Using the prepared methods

You can overload the following methods:

- SubscribeTo
- SubscribeToAllDocuments
- SubscribeToAllObjects
- SubscribeToDocuments
- SubscribeToObjects

Please see the IntelliSense tooltips which will help you decide which of the overloaded methods suits you best.

```
SubscribeToObjects(TaskProcessTypeEnum.AsyncSnapshot, PredefinedObjectType.USER);
```

This example show the easy way of subscribing to changes made to user objects.

Methods to be implemented

The following methods need to be implemented in the connector to ensure synchronization in the outbound direction:

- ProcessInternalTaskAsync - ensures asynchronous processing of objects or documents (depending on the chosen override).
- GetExternalObjectID - used in asynchronous processing for ID translations when synchronized objects or documents reference objects inheriting from BaseInfo.
- GetExternalDocumentID - used in asynchronous processing for ID translations when synchronized objects or documents reference documents (TreeNode).
- ProcessInternalTaskSync - ensures synchronous processing of objects or documents (depending on the chosen override).

ProcessInternalTaskAsync method

The method has two overrides. The first one is used for asynchronous processing of objects, the second one for asynchronous processing of documents.

```
IntegrationProcessResultEnum ProcessInternalTaskAsync(GeneralizedInfo infoObj,
TranslationHelper translations, TaskTypeEnum taskType, TaskDataTypeEnum dataType,
string siteName, out string errorMessage)

IntegrationProcessResultEnum ProcessInternalTaskAsync(TreeNode node,
TranslationHelper translations, TaskTypeEnum taskType, TaskDataTypeEnum dataType,
string siteName, out string errorMessage)
```

This method is used for asynchronous processing of objects (like users, forums, bad words, etc.) or documents. If we consider a basic scenario, your task is to transform the *GeneralizedInfo* or TreeNode into a corresponding object in the third party system and perform the action specified by TaskTypeEnum. You also have to take into account the TaskDataTypeEnum and the necessity of performing foreign key translations.

When you're done with processing, return the IntegrationProcessResultEnum value and eventually set some error message. The error message is visible through the UI.

ProcessInternalTaskSync method

The method has two overrides. The first one is used for synchronous processing of objects, the second one for synchronous processing of documents.

```
IntegrationProcessResultEnum ProcessInternalTaskSync(GeneralizedInfo infoObj,
TaskTypeEnum taskType, string siteName, out string errorMessage)

IntegrationProcessResultEnum ProcessInternalTaskSync(TreeNode node, TaskTypeEnum
taskType, string siteName, out string errorMessage)
```

These two methods work similarly as the asynchronous version *ProcessInternalTaskAsync()*. The only difference is that you are not given the opportunity to use the *TranslateColumnsToExternal()* method for the foreign key translations. If you want to translate column values, you have to use Kentico CMS API.

The following example shows approximate code used for translation of a node's parent identifier:

```
TreeProvider tree = new TreeProvider(CMSContext.CurrentUser);
TreeNode parentNode = DocumentHelper.GetDocument(node.NodeParentID, tree);
Guid parentGuid = parentNode.NodeGUID;
string parentSiteName = parentNode.NodeSiteName;

int newParentId = 0;

// TODO: External code which utilizes parentGuid and parentSiteName to find
corresponding external document and returns its parent identifier (newParentId)
```

```
node.NodeParentID = newParentId;
```

Task processing options

There are several options how processing of a task can be achieved to synchronize the data:

- Utilize API of the external system if you add references to its namespaces.
- Use *CMSConnectionScope* and *GeneralConnection* and perform a query upon the external database.
- Push the data to an external endpoint in a format which it is able to consume. This endpoint can be represented e.g. by a web service existing in the external system.

You can probably think of some other ways, but you should keep in mind that all *ProcessInternalTaskXXX* methods have to return the result status so that you are able to determine whether the processing on the external side succeeded or not. Please see Important types -> Enumerations -> IntegrationProcessResultEnum for further details.

Translating foreign key values to match the external ones

If you chose to use the *SimpleSnapshot* or *Snapshot* data type, you probably did it because you want to take advantage of the possibility to translate column values. This can be achieved by calling *TranslateColumnsToExternal()*, which accepts either an object or a document:

```
TranslateColumnsToExternal(infoObj, translations, <bool processChildren>)
```

The last parameter says whether you want to translate foreign keys of child objects. For the *SimpleSnapshot* data type, you always pass *false*. This parameter is useful only when you use the *Snapshot* data type. E.g. when you are processing an object that does not exist on the target platform yet, you don't have enough information to translate foreign keys of child objects and you need to process the main object first. So the recommended order of events is the following:

1) Call *TranslateColumnsToExternal(infoObj, translations, **false**)*.
2) Process the main object (save it to database).
3) Call *TranslateColumnsToExternal(infoObj, translations, **true**)*.
4) Iterate through Children collection of *infoObj* and process each object.

To ensure the method's proper functionality, it is necessary to implement one or both of the following methods:

- **GetExternalObjectID()** – use if synchronized objects or documents reference objects inheriting from BaseInfo.
- **GetExternalDocumentID()** – use if synchronized objects or documents reference documents (TreeNode).

GetExternalObjectID method

The method has the following signature:

```
int GetExternalObjectID(string objectType, string codeName, string siteName,
string parentType, int parentId, int groupId)
```

The method has to be implemented only when you use *TranslateColumnsToExternal()* upon an object or document referencing an object inheriting from *BaseInfo*. You are given a set of parameters which should help you to identify the corresponding object in the external system. Once you find it you just return its identifier (integer value).

Main parameters:

- objectType – defines type of object (e.g. "cms.user"), it can match external objects such as "people" or "members"
- codename – this is the main identifier

Additional parameters:

- siteName – in case the object belongs to some site, this is its code name
- parentType – if the object has any parent, this is its type
- parentId – if the object has any parent, this is its identifier
- groupId – if the object belongs to some group, this is its identifier

GetExternalDocumentID method

The method has the following signature:

```
int GetExternalDocumentID(Guid nodeGuid, string cultureCode, string siteName, bool
returnDocumentId)
```

The method has to be implemented only when you use *TranslateColumnsToExternal()* upon an object or document referencing a document (*TreeNode*). You are given a set of parameters which should help you identify the corresponding document in the external system. Once you find it, you just return its identifier (integer value).

Parameters:

- nodeGuid – guid part of the identifier.
- cultureCode – the culture code of the document (e.g. "en-US").
- siteName – document in Kentico CMS is always bound to some site therefore the site code name is also supplied.
- returnDocumentId – determines whether the system requires you to return *DocumentID* or *NodeID* value. *NodeID* is identifier of document independent on culture while *DocumentID* identifies specific culture version of a document. See the table below for more details.

	Identifier	Alternative identification
Shared document data	NodeID	nodeGuid & siteName
Culture version of a document	DocumentID	nodeGuide & siteName & cultureCode

For more information on how Kentico CMS documents are stored in the database, please refer to Developer's Guide -> Content management -> Content management internals and API -> Database tables.

Translation flow diagram

The following figure illustrates how the translation works:
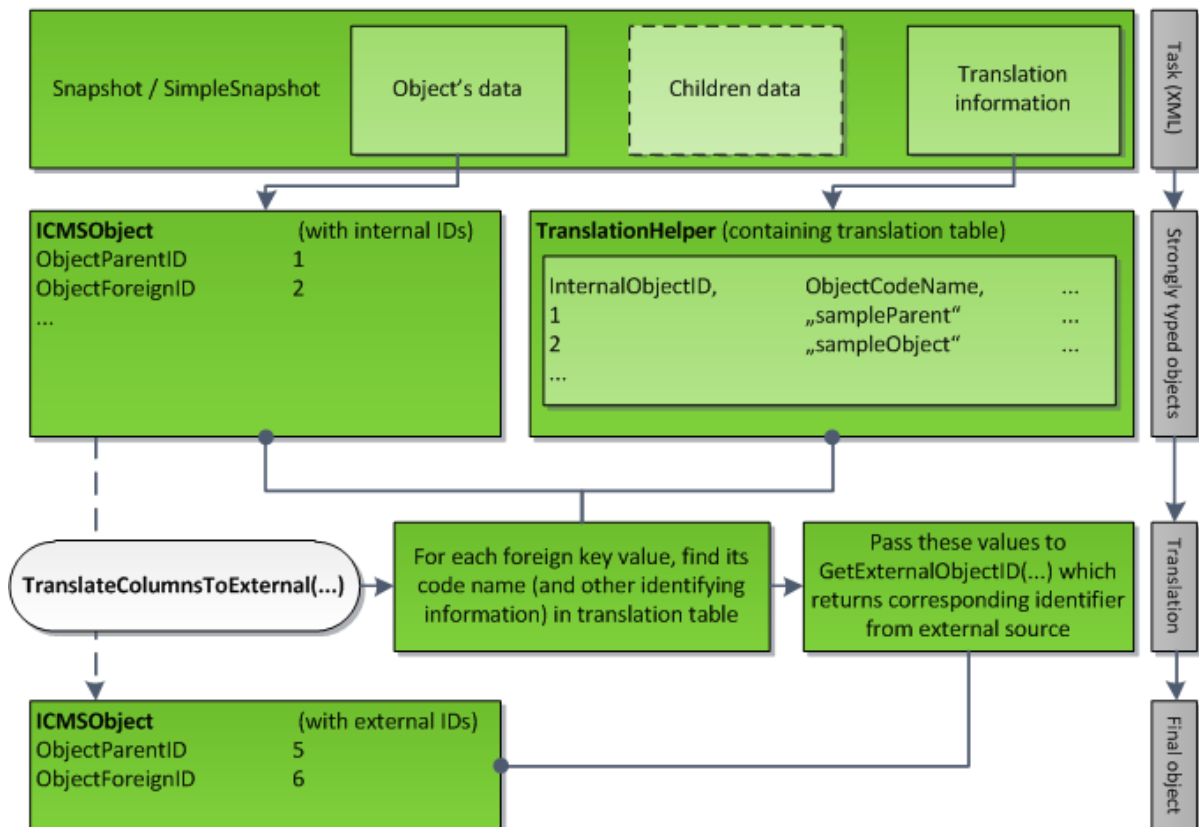
4.3   Implementation of inbound direction

At this point you should be decided:

- which objects and documents you want to synchronize
- which data type you want to use

Based on this information you will be able to decide which methods you need to implement. You should also have operational skeleton of connector prepared as described in the Creating a connector class topic.

Implementation basics

To log object or document tasks to the queue, please use the following method located in the *IntegrationHelper* class:

```
void ProcessExternalTask(string connectorName, object obj,
IntegrationProcessTypeEnum result, TaskTypeEnum taskType, TaskDataTypeEnum
dataType, string siteName)
```

This method just logs the tasks to the queue and the system takes care of them and processes them later. It has the following parameters:

- **connectorName** – use code name of the connector for that you want the tasks to be logged.
- **obj** – this will typically be the external object. If you have somehow managed to prepare *ICMSObject* earlier, you can pass it as well.
- **result** – this value says how the system should behave when fetching the tasks from the database and how it should react if an error occurs.
- **taskType** – by providing this value, you say whether the provided object or document should be created, updated, deleted, etc.
- **dataType** – says whether the provided object contains also child objects, etc. The value also indicates whether the methods for collecting translation information will be called.
- **siteName** – if the processed object belongs to a site, you should provide a code name of this site.

Detailed description of particular enumerations used in the parameters can be found in Important types -> Enumerations.

The following diagram illustrates the sequence of method calls for each inbound synchronization. The purpose and details of individual methods are described further below.
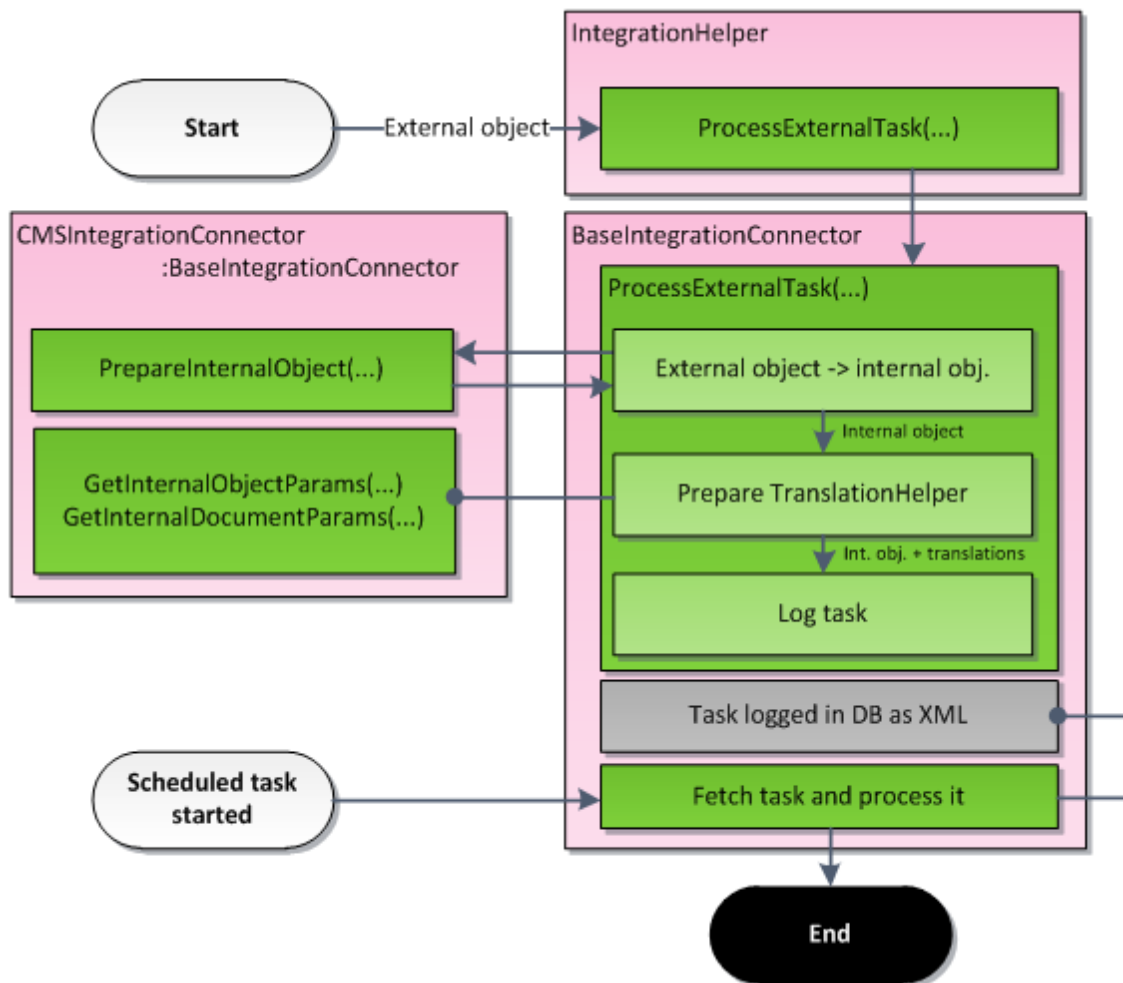
Methods to be implemented

The following methods need to be implemented to ensure synchronization in the inbound direction:

- PrepareInternalObject - ensures processing of an object or a document.
- GetInteralObjectParams - prepares translation information for an object or a document that has a foreign key to an object (BaseInfo).
- GetInternalDocumentParams - prepares translation information for an object or a document that has a foreign key to a document (TreeNode).

PrepareInternalObject method

The method has the following signature:

```
ICMSObject PrepareInternalObject(object obj, TaskTypeEnum taskType,
TaskDataTypeEnum dataType, string siteName)
```

The method serves as a centralized point for transformation of objects and documents from the external system to the corresponding ones in Kentico CMS. Your task is to return a valid TreeNode or object

inheriting from BaseInfo. The following figure illustrates when the method is called:

GetInernalObjectParams method

The method has the following signature:
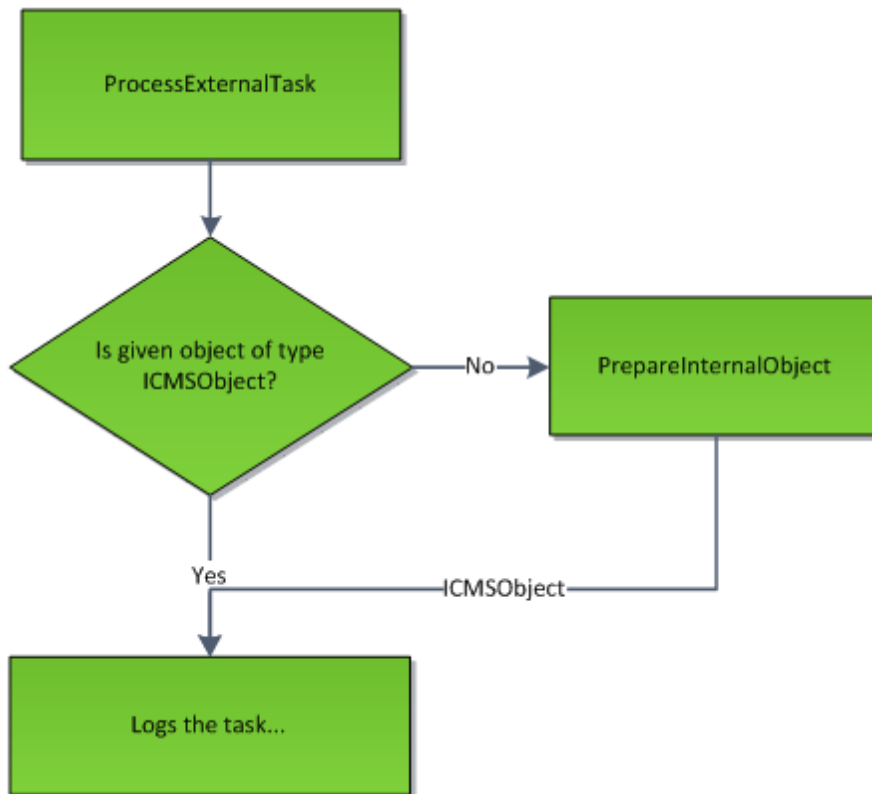
```
void GetInternalObjectParams(int id, string objectType, out string codeName, out
string siteName, ref int parentId, ref int groupId)
```

The method needs to be implemented when you are planning to synchronize objects or documents that have foreign keys referencing objects (in Kentico CMS). Based on the given parameters (*id* and *objectType*), you should be able to find corresponding object in the external system and supply at least its code name through the corresponding *out* parameter. It is possible to specify the object more precisely by providing also *siteName*, *parentId* and *groupId*.

GetInternalDocumentParams method

The method has the following signature:

```
void GetInternalDocumentParams(int id, string className, out Guid nodeGuid, out
string cultureCode, out string siteName)
```

The method needs to be implemented when you are planning to synchronize objects or documents that

have foreign keys referencing documents (in Kentico CMS). Based on the given parameters (*id* and *className*), you should be able to find the corresponding document in the external system and supply its *nodeGuid*, *cultureCode* and *siteName* through the *out* parameters. All of the parameters are mandatory.

How to request processing of logged tasks

BaseIntegrationConnector offers you two overloads of the *RequestTasksProcessing* method:

```
HttpStatusCode RequestTasksProcessing(string serverUrl)
HttpStatusCode RequestTasksProcessing(string serverUrl, string connectorName)
```

By calling these methods, the application makes a HTTP request to the *~/CMSPages/ IntegrationNotify.aspx* page, which causes that processing is executed. Each connector will be processed in its own thread. The method requires you to specify a URL leading to the root of the Kentico CMS application (e.g. *http://www.example.com/KenticoCMS*). The second overload allows you to specify the connector whose tasks will be processed. If the parameter is not supplied, all connectors will be processed.

Part

V
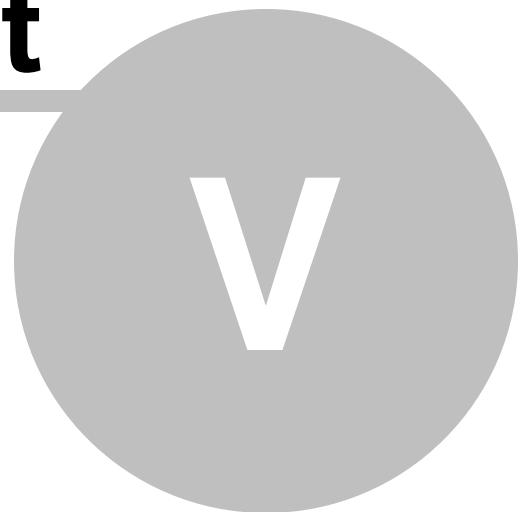
Advanced scenarios

5   Advanced scenarios

5.1   Creating a custom subscription class

You can create your own subscription class by implementing the *IsMatch()* method. This can be useful when you want to extend the options of defining the scope.

The diagram below illustrates inheritance of existing classes:
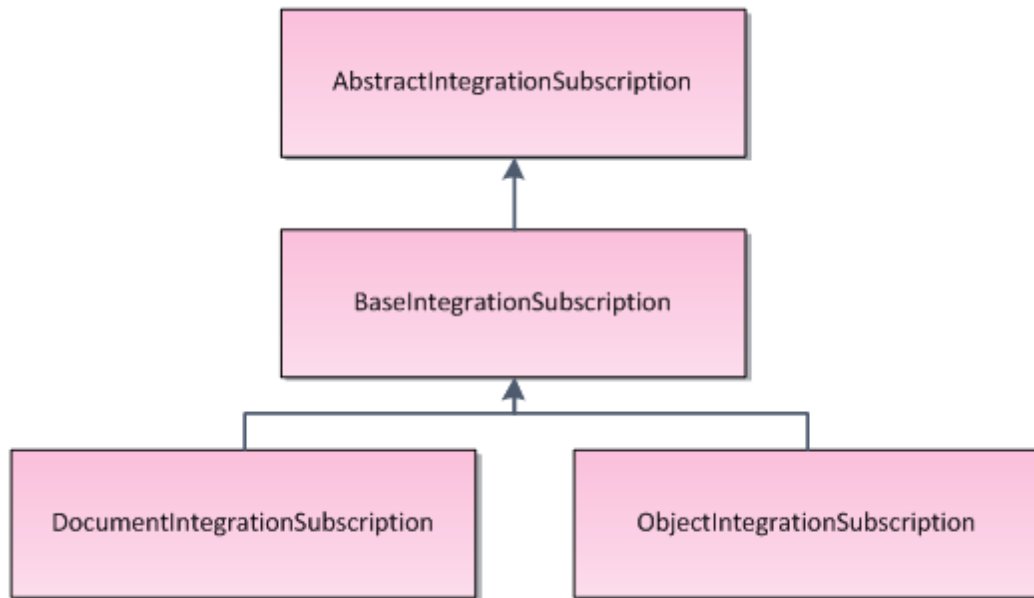
Choose one and inherit your subscription from it. Now you have to implement an override of the following method:

```
bool IsMatch(ICMSObject obj, TaskTypeEnum taskType, ref TaskProcessTypeEnum
taskProcessType);
```

Your task is to evaluate whether the subscription or more precisely its properties (initialized in constructor) match the properties of *obj* and the value of *taskType*. Once they do, you simply initialize *taskProcessType* (which should be also initialized in constructor of subscription) and return *true*.

Part VI

Important types

6   Important types

This chapter gives you general overview of data types that you can be confronted with during implementation.

Classes

- **BaseInfo** - represents any site-related or global data object in Kentico CMS (e.g. a page template or a poll).
- **TreeNode** - represents a document in Kentico CMS.

Interfaces

- **ICMSObject** - interface identifying *BaseInfo* or *TreeNode*.

Enumerations

TaskTypeEnum

This enumeration is located in the *CMS.SettingsProvider* namespace. It always accompanies a processed object or document. The value has a slightly different meaning for outgoing and incoming tasks:

- for outgoing tasks, the value is used only to create subscriptions and it says which actions should be handled. E.g. when you subscribe to *CreateObject*, the Integration bus will react only when object inheriting from BaseInfo is created.
- for incoming task, it says what should happen with the given object. E.g. when you pass a document object and task type *DeleteDocument*, Kentico CMS will delete the document.

The most common values for objects are:

- CreateObject
- UpdateObject
- DeleteObject
- AddToSite (not applicable for global objects)
- RemoveFromSite (not applicable for global objects)
- All - this value gives sense only for outbound direction (during subscribing). It indicates that integration bus should create tasks whenever change to document or object is made.

The most common values of documents are:

- CreateDocument
- UpdateDocument
- DeleteDocument
- PublishDocument
- ArchiveDocument
- All – this value gives sense only for outbound direction (during subscribing). It indicates that integration bus should create tasks whenever change to document or object is made.

To see the whole list of values, please explore the enumeration in Visual Studio. You can use object browser if you don't have source code.

Please note that there is no task type indicating a change of documents' worklow steps (except special types like *PublishDocument* or *ArchiveDocument*). This is one of the known limitations which are likely to be fixed in one of the future versions.

TaskProcessTypeEnum

Values of this enumeration represent all supported combinations of synchronicity and data modes. Please note that when the system looks for connectors that have a matching subscription and one connector has more of them, the system takes the first one according to the following priority (highest to lowest):

- SyncSnapshot
- AsyncSnapshot
- AsyncSimpleSnapshot
- AsyncSimple

As you can notice, there is only one *Sync* option — SyncSnapshot. This is given by the nature of synchronous processing where you can always access related objects (parent, children, etc.).

TaskDataTypeEnum

You can notice the usage of *TaskDataTypeEnum* across whole integration module. This enumeration gives additional information about the content of the currently synchronized object. For outbound direction, the value also specifies whether the object is being accompanied with the translation data — this applies only to asynchronous processing. For inbound direction, it says whether the translations should be prepared during task logging and whether the system should try to process child objects. Translation data are data enabling translation of foreign keys between Kentico CMS and the external system. When using synchronous processing, the required data can be obtained directly within the context of the application.

There are three options:

Value	Object data	Translation data (only for asynchronous processing)	Include child objects, bindings, categories, etc.
Snapshot	✔	✔	✔
SimpleSnapshot	✔	✔	
Simple	✔		

Snapshot mode does not support documents

Please note that the **Snapshot** mode is not supported for synchronization of documents. This is one of the limitations that will be fixed in one of the future Kentico CMS versions.

IntegrationProcessResultEnum

This enumeration is used for indicating the result of processing outgoing task by 3rd party applications. The result determines what happens after processing a single task.

Possible values are:

Value	Meaning	Asynchronous processing	Synchronous processing
OK	Processing succeeded	Task (or relation between task and connector) is deleted.Processing continues with the next task in queue.	Processing continues with the next task in queue.
Error	Critical error occurred	Error is logged to the synchronization log. Processing stops for the current connector.	Error is logged to event log. Task data are lost. Processing stops for all connectors.
ErrorAndSkip	Noncritical error occurred	Error is logged to the synchronization log. Processing continues.	Error is logged to the event log. Task data are lost. Processing stops for the current connector.
SkipNow	Process the task during next iteration	Processing continues with the next task in queue.	Task data are lost.

IntegrationProcessTypeEnum

Values of this enumeration are used during processing of incoming tasks. Generally, it says whether to process the task immediately or not and how to behave when an error occurs.

Value	Meaning
Default	Processes the task immediately. If an error occurs, the processing stops and the type is set to *Error*.
SkipOnce	Does not process the task during the first processing (just sets the type to *Default* so it is going to be processed during next processing).
SkipOnError	Processes the task immediately. If an error occurs, the task is skipped and the processing continues.
DeleteOnError	Processes the task immediately. If an error occurs, the task is deleted and the processing continues.
Error	Processing should not continue due to a critical error.

Part

VII

Database model

7   Database model

The Integration bus module uses the following database tables:

- **Integration_Connector** – contains definitions of connectors. The system therefore knows which connectors should be loaded and from which assemblies.
- **Integration_Task** – contains data of outgoing and incoming tasks. It also contains additional information like TaskDataType or TaskProcessType.
- **Integration_Synchronization** – contains relations between tasks and connectors. For incoming tasks, there is always one record for each task. For outgoing tasks, there can be multiple records for each task — this depends on whether there are some connectors with the same subscription. This design saves database storage capacity.
- **Integration_SyncLog** – contains possible errors. Generally speaking, when task processing fails, the system stores the error messages here.