MEMOQ SERVER CMS GATEWAY API

API AND SDK DOCUMENTATION

LEGAL

Author: memoQ Translation Technologies

Copyright notice: Produced by (P) memoQ, 2004–2018. This document is the intellectual property of memoQ Translation Technologies Ltd. Reproduction of this document, either in whole or in parts, and including but not limited to analogue and electronic copying, distribution and storage is allowed only to the extent described by respective agreements or upon prior written permission issued by memoQ.

DOCUMENT HISTORY

VERSION	TYPE	DATE	AUTHOR	CHANGES
1.00	Working	2017-07-06	Gábor Nagy	API reference.
1.01	Working	2017-07-10	Gábor Nagy	Authentication, location header
1.02	Working	2017-07-22	Zoltán Benedek	Client name and e-mail is returned. List of languages. List of language pairs. Language pair related error codes.
1.03	Working	2017-08-22	Zsolt Paral	New notification system
1.04	Published	2018-11-23	Gusztáv Jánvári	Adding conceptual and logical design. Adding sample implementation's documentation. FAQ.

TABLE OF CONTENTS

1. ABOUT

This document describes how CMS Connectors can be integrated with memoQ Server's CMS Gateway using its CMS API. The API is based on REST technology.

A NOTE ON FIGURES

The document may contain large diagrams and details may be hard to catch. Feel free to zoom into the relevant pages, all such figures are vector graphics.

1.1. STRUCTURE OF THE DOCUMENT

Chapter 2, Conceptual Design introduces you to the world of memoQ CMS integration by presenting the system landscape, the key entities involved in the collaborations and the workflows that can be facilitated with such integrations.

Workflows consist of les and more complex activities or, in other terms, operations and collaborations. Chapter 3, Collaboration Protocols, as a logical design, takes the collaborations defined in the workflows, and presents the sequence of message exchanges to perform in order to perform a collaboration.

The description of operations involved in the collaborations refer to the description of separate requests, collected under chapter 4, API Reference, as a means of implementation-level design.

The document is completed by a set of Frequently Asked Questions in chapter 6.

1.2. RELATED STUFF

The documentation is accompanied by a sample CMS Connector implementation, discussed in chapter 5, Sample CMS Connector Application.

1.3. UPDATES

Please note that this documentation, the sample implementation and memoQ Server are all evolving products and may change time to time. It is recommended to regularly check if there is a newer version of this API documentation or the sample implementation on memoQ's website memoq.com.

2. CONCEPTUAL DESIGN

2.1. SYSTEM LANDSCAPE

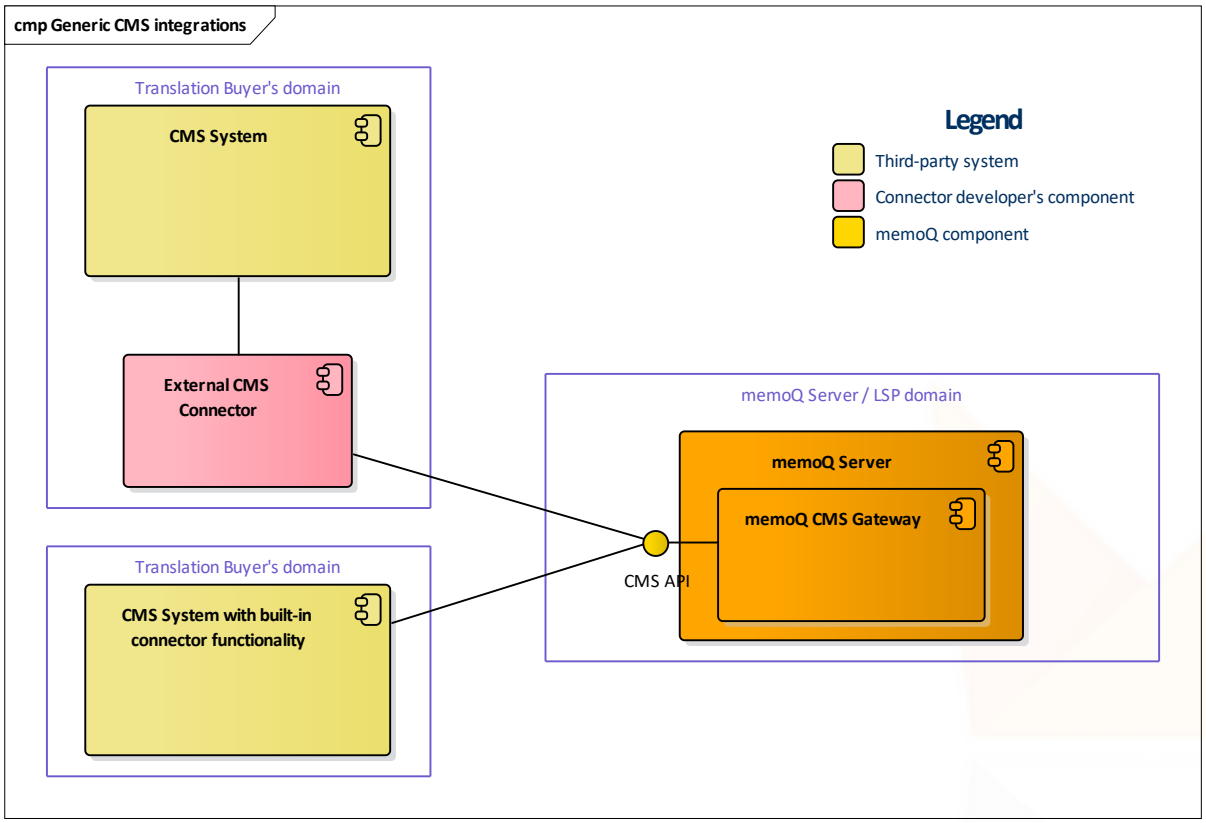
The **CMS Gateway** is a component of **memoQ Server**. The Gateway exposes an API, called the **CMS API**, which can be used by third-party applications, called **CMS Connectors**, to facilitate content transfer between third-party systems (CMS systems, proprietary systems or data-

bases, or file systems; collectively referred to as **CMS systems**) and memoQ Server. The purpose of this content transfer is to **submit translatable contents** to memoQ Server and **download translated content** from memoQ Server. Unless the CMS Connector functionality is built into the CMS System, it is the CMS Connector's responsibility to obtain translatable contents from the CMS system and deliver translated contents to that CMS system.

NOTES:

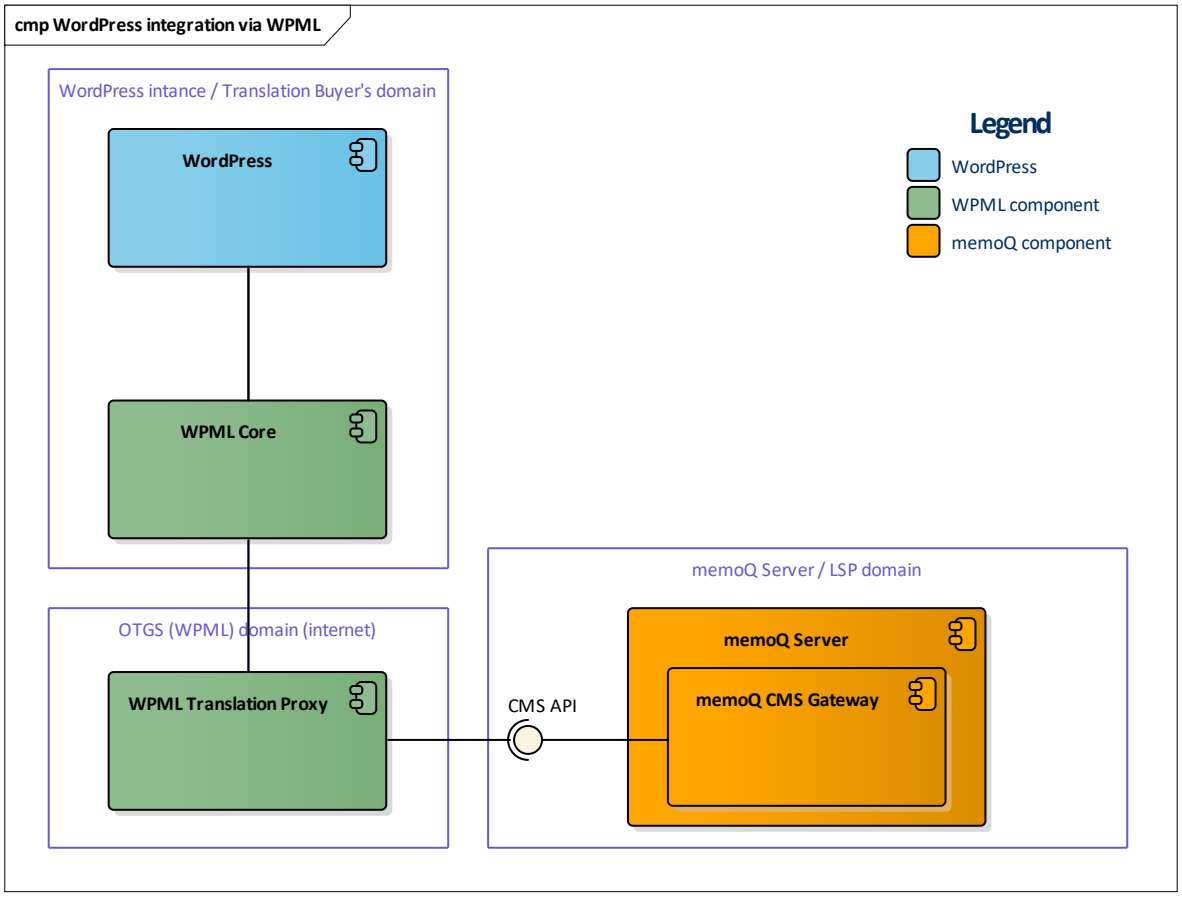
- Some third-party systems may be enhanced to connect to the CMS API, some may need an external connector. It is up to the developer party whether the CMS API client is implemented as part of the third-party system (referred to as built-in connector functionality) or as an external connector.
- memoQ does not contain any built-in connectors and will likely not ship any with memoQ.
- OnTheGoSystems, the vendor of the WordPress localization platform WPML has developed a connector to WordPress via WPML. This can be used to connect WordPress sites to memoQ Server via OnTheGoSystems, and to automate the localization of WordPress sites.

Figure 1. Architecture of CMS integration solutions



For the sake of the reference and the example, the situation with the existing WordPress integration is somewhat different in that the external CMS connector component is implemented in the middleware of its vendor, so the architecture looks like this:

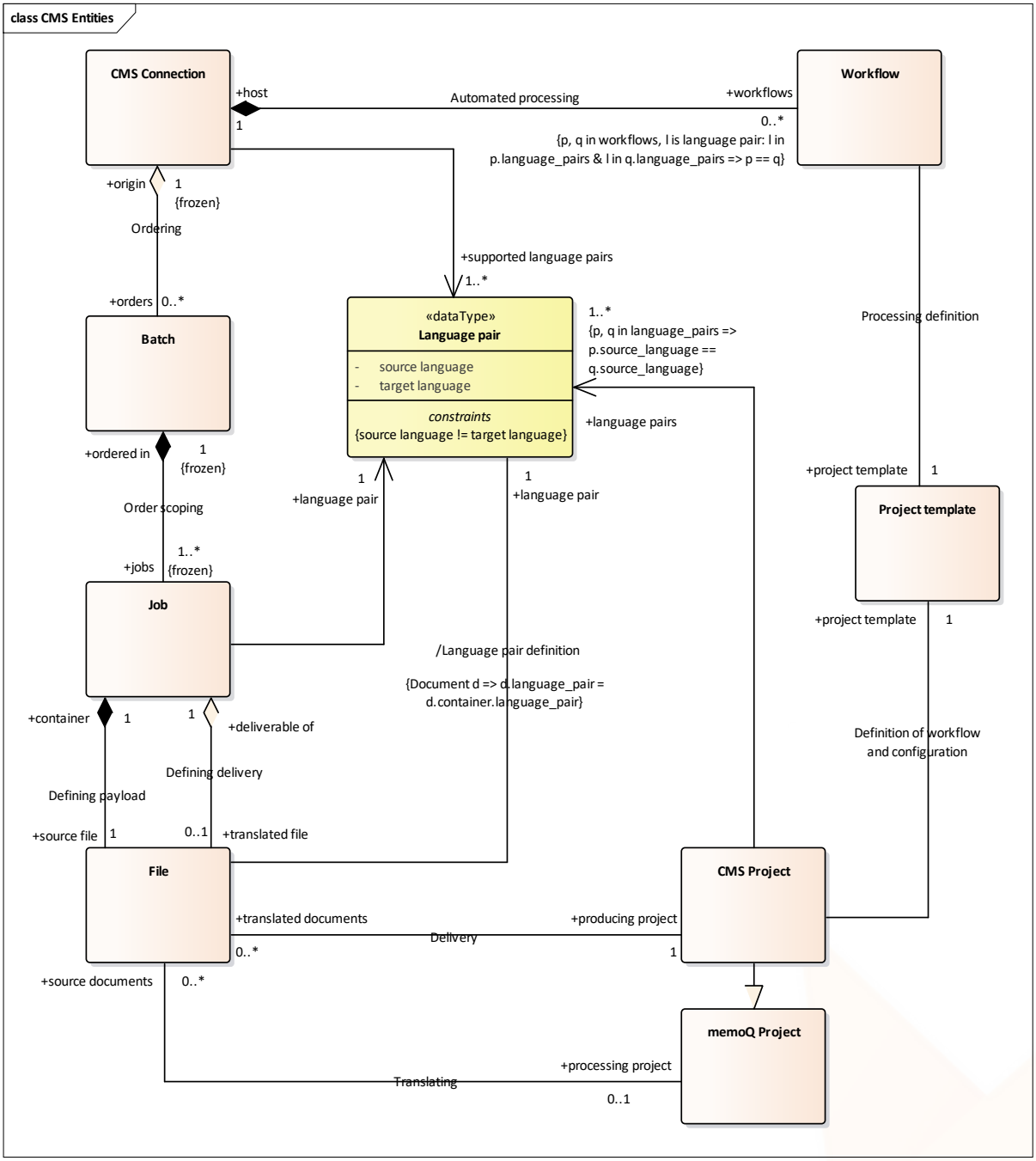
Figure 2. Architectural view of WordPress-memoQ integration via WPML Translation Proxy



2.2. ENTITIES

This chapter outlines the key entities related to the CMS Gateway and API to provide a common understanding of the underlying concepts and processes. First let's take a look at the domain model, explained below the diagram.

Figure 3. CMS domain model



When a specific memoQ Server and a CMS system of a The Translation Buyer is integrated, a so-called **CMS Connection** is created, which contains information about that connections, such as the supported languages or the end-customer's name. After a connection is established, the Translation Buyer can submit translation **orders** in the form of **batches** to memoQ Server. Batches do not survive connections: if a connection is deleted, it's existing batches are deleted as well. Note that we currently have some ambiguity in our terminology: in the API level, batches are called projects. This will likely be changed in the future, and a new term, order will be introduced to better reflect the purpose of this construct.

Batches contain **jobs**, defined and submitted by the CMS Connector to memoQ. Jobs are task containers, consisting of some metadata like source and target language, deadline, file type,

and a **source file** to translate (the payload), and the **translated file**. This translated file shall be downloaded by the CMS Connector.

Jobs received from CMS Connections are processed in **CMS projects**. A CMS project is similar to regular memoQ projects, with some key exceptions:

- CMS projects can only contain files defined in CMS jobs.
- Delivery to the CMS system can be initiated for files in CMS projects only.

Users can define **workflows** for CMS Connections. Workflows define a set of source and target language pairs and a **project template**. When a new batch of jobs is received over that particular CMS connection, the source files of jobs matching the language pairs supported by the workflow will automatically be processed by the attached project template.

The following rules apply for source and target languages:

- A CMS connection can support any language pairs. This can be defined by the memoQ Server Administrator for each connection.
- Each job has one source and one target language specified. Batches may contain jobs with different language pairs.
- A CMS project can only have one source language, similarly to other memoQ projects, and support any number of target languages.
- Workflows support specific language pairs, and these form a subset of the language pairs supported by the corresponding CMS Connection.
- Each particular language pair can only be supported by one workflow within a CMS connection.
- The source and target language in a language pair can never be the same—similarly to other parts of memoQ.

2.2.1. JOBS

The key component of the domain model is the CMS job, or job, for short. Jobs are the assignments the end customer (the user of the connected system) submits to the service provider.

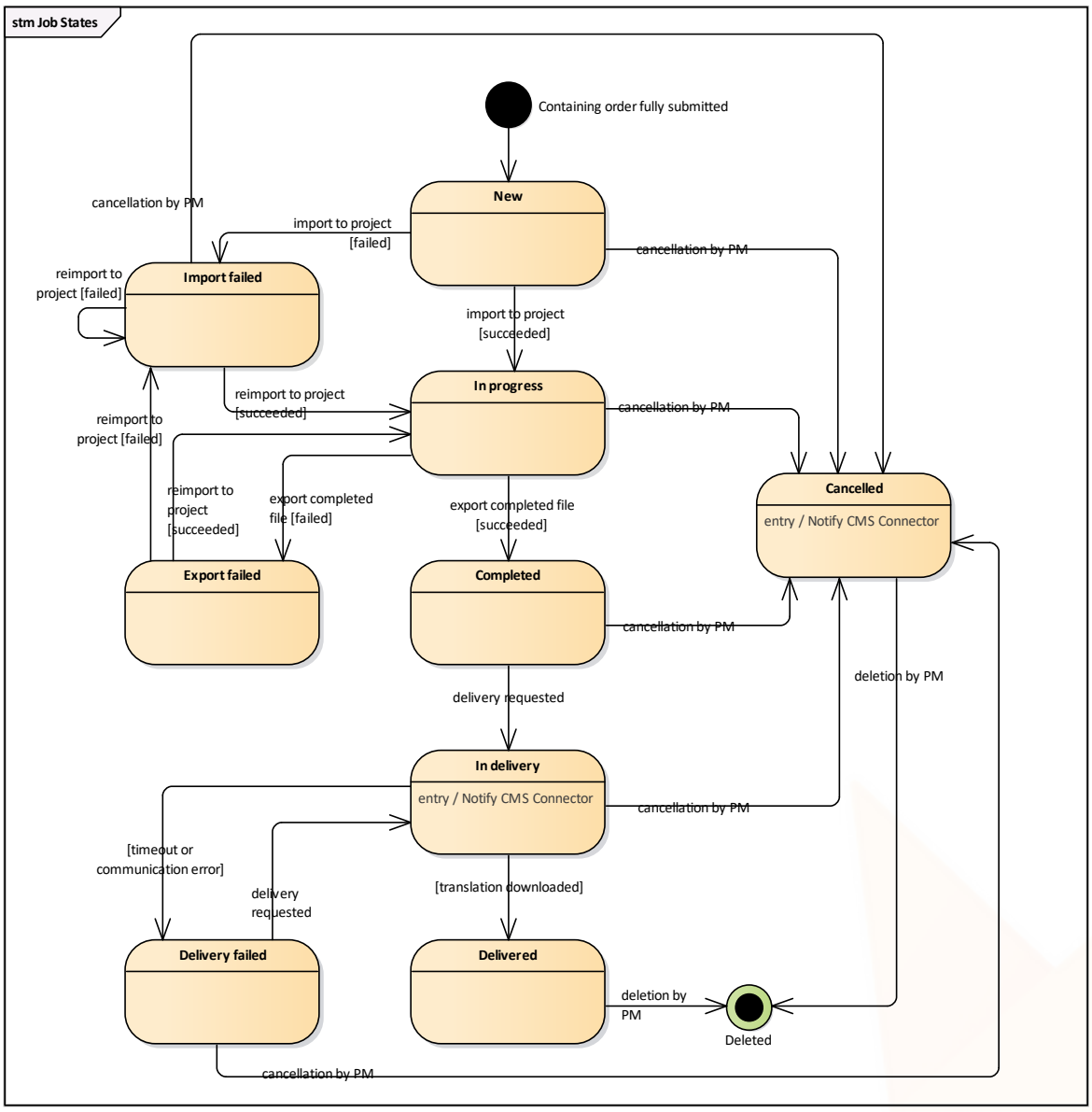
NOTE: Whether or not such an assignment means actually ordering the translation depends on the provider. They can set up memoQ in a way that new jobs are automatically pushed into the translation workflow, but they can also set it up in a way that only analysis is performed on the jobs, and then the parties can negotiate out of this integration solution whether or not the job is actually ordered.

CMS jobs have some relevant properties, some of them mentioned in the preceding section:

- One source and one target language
- The file containing the contents to translate (the translatable)
- The file containing the translated content (the deliverable)
- The specification of the content type (file type)
- Deadline – note that memoQ currently disregards the deadline

Practically the translation service provided by the provider to the user of the connected system is organized around jobs: jobs are received, processed and delivered. The whole system, however, is a bit more complex, and although much of a job's life is internal to memoQ, integrator parties may have a better understanding of the integration after learning the life of these jobs. This life is summarized in the following state machine.

Figure 4. CMS Job states



KEY NOTES

- Jobs can be processed manually and automatically on the memoQ side.
- Jobs can be delivered manually and automatically on the memoQ side.
- Jobs of a memoQ project can be delivered at once, or one by one, as each file completes its workflow within memoQ. In both cases, the CMS Connector needs to download deliverables one by one.

- memoQ PMs can import the same job's file as many times as they want: they can perform reimport if they realize the initial import had unwanted effects; or they can add a translatable to a different project for whatever reason.
- memoQ PMs can even import delivered jobs to memoQ projects. If it turns out a deliverable has problems that can be remedied in memoQ (by importing the file with a different filter, or by fixing some translation or QA errors in the files), they can do so by reimporting the affected jobs.
- memoQ PMs can deliver a job multiple times. In case of a re-delivery the CMS Connector shall assume there's a good reason for that, such as the affected deliverable is fixed, and shall download the deliverable even if they have downloaded a previous version earlier.
- A job can be cancelled any time until it is not delivered. memoQ notifies the CMS Connector about the cancellation of each job.
- The only other case when memoQ starts to talk to the CMS Connector is when delivery of a job is initiated automatically or manually. Then the CMS Connector can request the deliverable in a message.

2.3. WORKFLOWS

After discussing the key entities, let's see how CMS connectors can interact with memoQ in order to have contents of external systems translated by memoQ. When it comes to CMS integration, we basically need to differentiate between two things, with their associated workflows:

- Establishing connection between memoQ and a third-party system. This is basically a one-off process for each CMS connection in memoQ. See CMS Connection Lifecycle for details.
- Translating content via memoQ. If we apply a black-box technique and don't pay attention to the internal operation of memoQ, this is one key workflow, which can be customized within memoQ, and involves the transportation of translatable contents to memoQ, the translation of contents, and the transportation of translated contents back to the CMS connector. This is detailed in Translation Workflow.

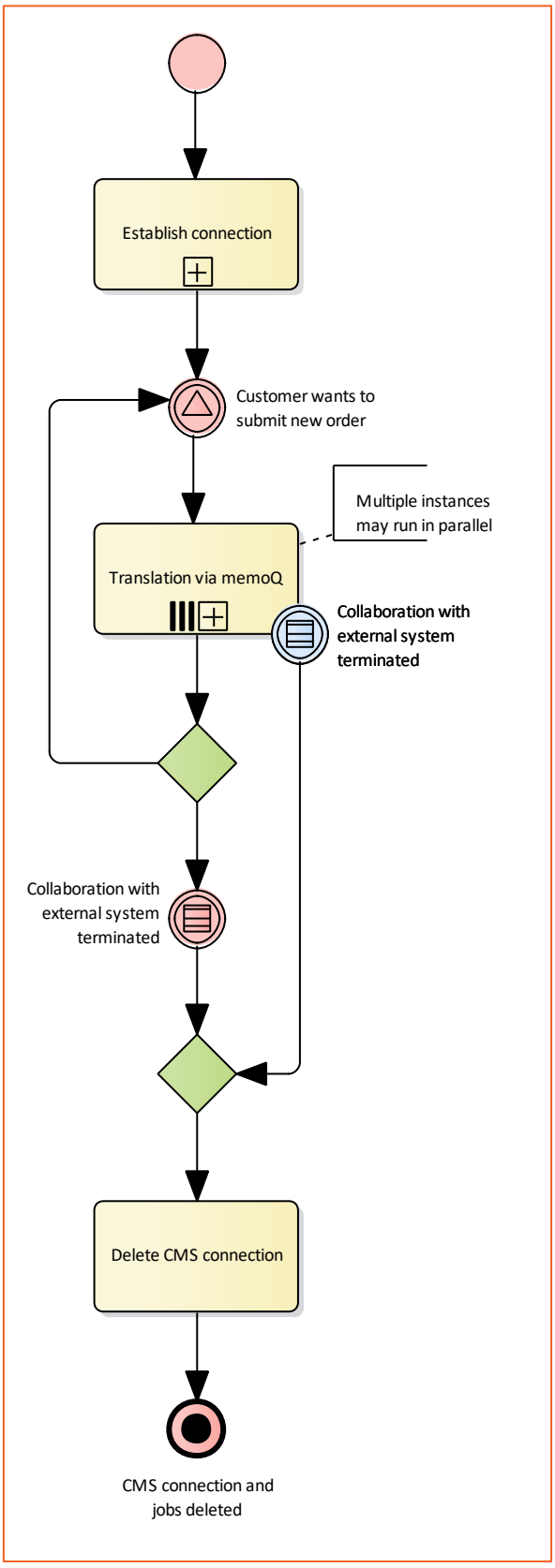
2.3.1. CMS CONNECTION LIFECYCLE

Before being able to submit contents from a CMS Connector to the CMS Gateway via the CMS API, the connection first needs to be created and registered on both sides. Creation of CMS connections on memoQ Server is subject to having a proper CMS license (the WordPress integration license) and to having the CMS Gateway enabled in memoQ Server's Deployment Tool.

After successfully establishing connection, content transfer can take place in both directions as long as the connector is not deleted, and the memoQ Server holds a proper CMS license.

The lifecycle of CMS connections is shown on the following BPMN business process diagram.

Figure 5. CMS Connection lifecycle workflow



KEY NOTES

- Once a CMS connection is established, translation of contents in memoQ can take place in a simultaneous manner, meaning that starting a new translation process does not need to wait for the end of another started earlier.
- Once there's no need for a CMS connection any longer, it can be deleted.
- If the connection information changes, the registration procedure Establish Connection needs to be repeated.
- The workflows of translating contents via memoQ is detailed in section Translation via memoQ.

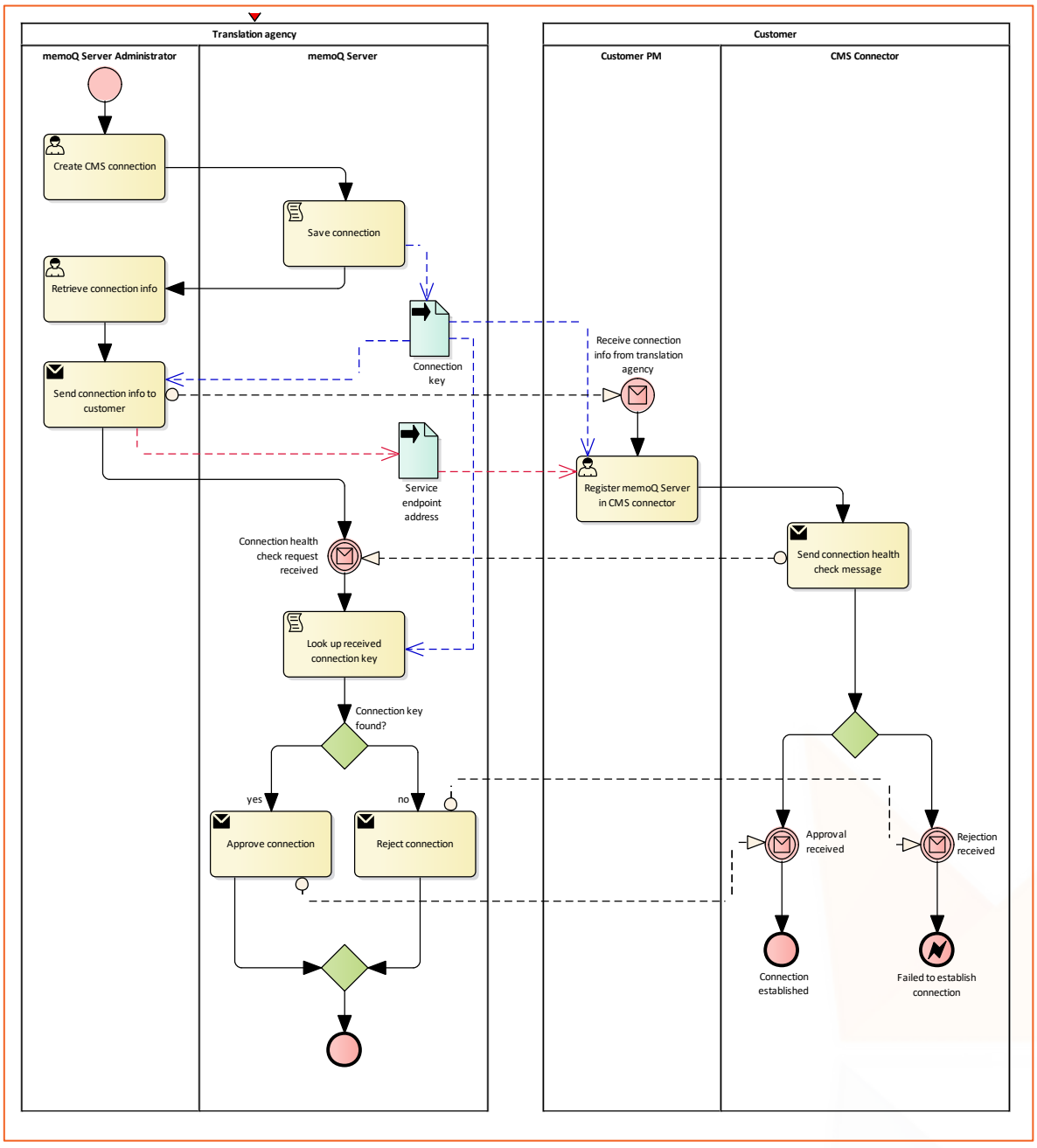
The following section details the operations related to CMS connections.

2.3.2. CMS CONNECTION MANAGEMENT

2.3.2/A ESTABLISH CONNECTION

Creating a CMS connection consists of a couple of manual and programmed steps, shown in the following figure.

Figure 6. Workflow of setting up a CMS connection



KEY NOTES

- The CMS connection first needs to be created in memoQ Server's **Server Administrator** dialog. As part of the creation process, a secret, called the **connection key** is created.
- The Server Administrator can then get all information required for the other side to connect to memoQ Server, and submit that information to the CMS Connector's authorized

user, called the **Customer PM**, in email, for example. The connection information contains, among others, the connection key and the **service endpoint address** of the CMS API.

- The Customer PM needs to register this specific memoQ Server instance in the CMS Connector by specifying these details. The CMS Connector may request for other information based on its sole decision.
- Once connection data is entered, the CMS Connector can (and is recommended to) check if the connection actually works by saying hello to the memoQ Server instance. To do so, it needs to submit its connection key to memoQ Server in a **connection health check** request.
- Based on whether memoQ Server can find a CMS connection with the connection key received in this message, it will either approve or reject the connection request in its response message.

2.3.2/B EDITING CMS CONNECTIONS

Existing CMS connections can be edited in memoQ. Editing only needs caution when you want to change the connection key. From the moment on you save a new connection key, the CMS connector needs to use the new connection key, or the API calls will be ignored. This may require the re-registration of the memoQ Server in the CMS connector, or editing the registration info in the CMS connector's UI, depending on how you implement this. We suggest having an option for replacing connection keys as a user-friendly effort to increase security by being able to change the secret time to time or in case of alleged compromise.

In terms of workflows and processes, basically the Establish Connection workflow needs to be repeated, only that no new records are created, but existing ones are modified.

2.3.2/C DELETING CMS CONNECTIONS

The CMS API does not have any requirements against CMS Connectors for connection deletion, that is, on its sole decision, CMS Connectors can delete connections and de-register memoQ Servers without sharing any information about this with that particular server instance. Note, however, if there are workflows in progress on memoQ Server, errors may occur when the CMS Gateway wants to notify the CMS Connector about some events.

CMS connections can be deleted on memoQ Server's Server Administrator UI. **Deleting a CMS connection irreversibly deletes all jobs ever transferred successfully to the CMS Gateway.** Deleting jobs deletes all the translation files received and translated files created (whether delivered or not) in the course of processing those jobs, however the translation files imported into a memoQ projects and their translations existing there are not deleted from memoQ, having a chance to recover from inadvertent deletions—they can be deleted by deleting the respective projects, and to store them for later re-use. The CMS Gateway does not notify the CMS Connector about the deletion, so CMS Connectors should handle cases of disappearing CMS Gateways and unavailable service endpoints (while the reason of such symptoms can be some intermittent network issue as well).

2.3.3. TRANSLATION VIA MEMOQ

TERMINOLOGY INCONSISTENCY NOTE. We currently struggle with some inherited terminology inconsistency here: **orders** are named **batches** in memoQ, and **projects** in the CMS API's world. For now, the three terms are interchangeable. Whenever possible, we'll use the term

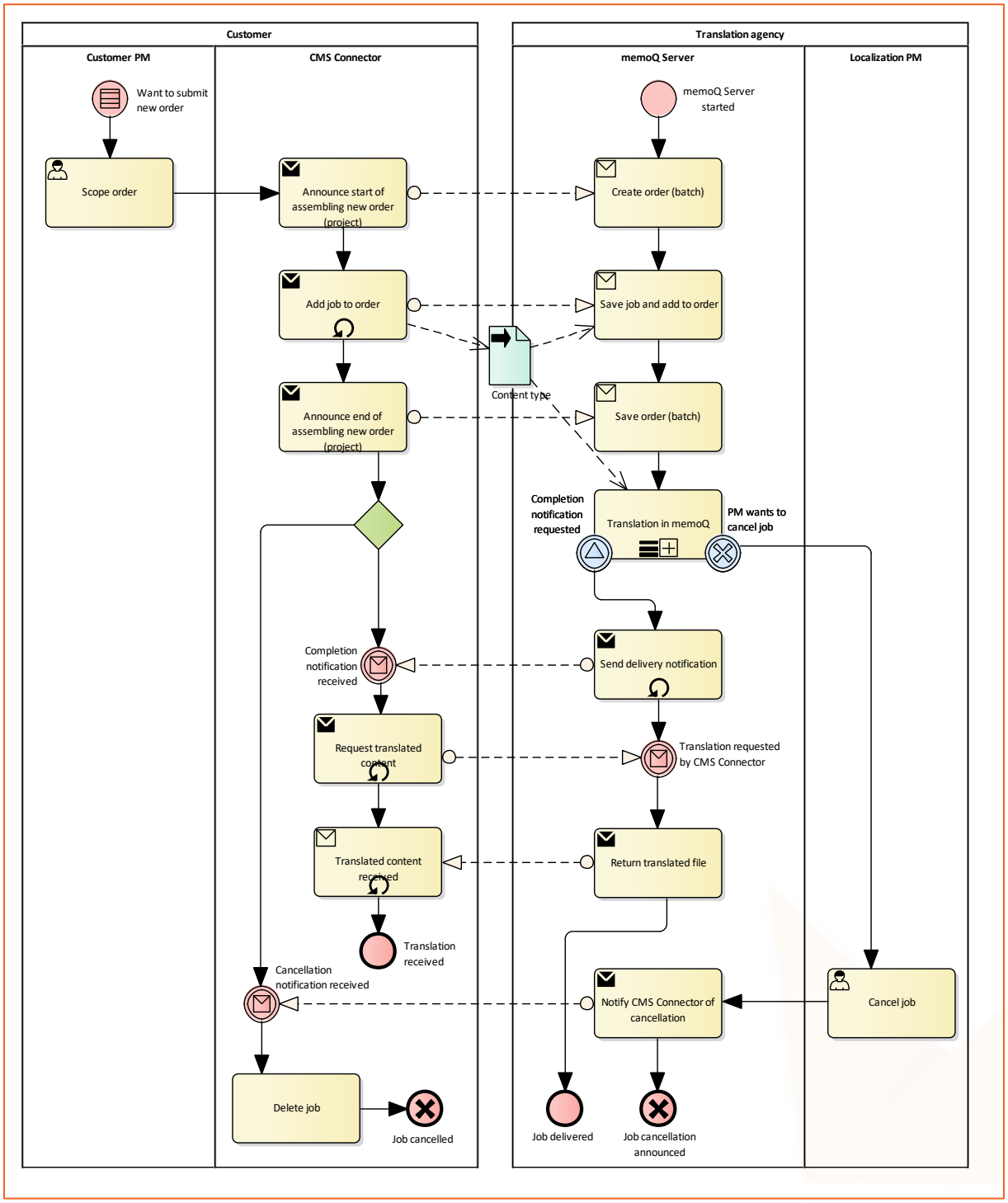
order. The following diagram, however, will use all the three terms to help you better understand the current terminology.

Speaking at the highest level:

- CMS jobs are submitted to memoQ Servers in orders. When an order is fully received, memoQ starts processing the order.
- When memoQ encounters events instructing memoQ to let the CMS Connector know a job is complete, it will notify the connector.
- The connector needs to download completed jobs with translated contents one by one.

The following diagram shows the workflow in more details.

Figure 7. Workflow of translation via memoQ



KEY NOTES

- The CMS Connector may provide a means for the customer to scope orders by, for example, letting them to select which assets to submit for localization as part of that particular order. It can, however, use automated solutions to detect new translatable contents. It's entirely up to the connector.
- CMS Connector need to start submitting orders by creating a new project on the API, which is called a batch in memoQ—remember the terminology inconsistency; in the future

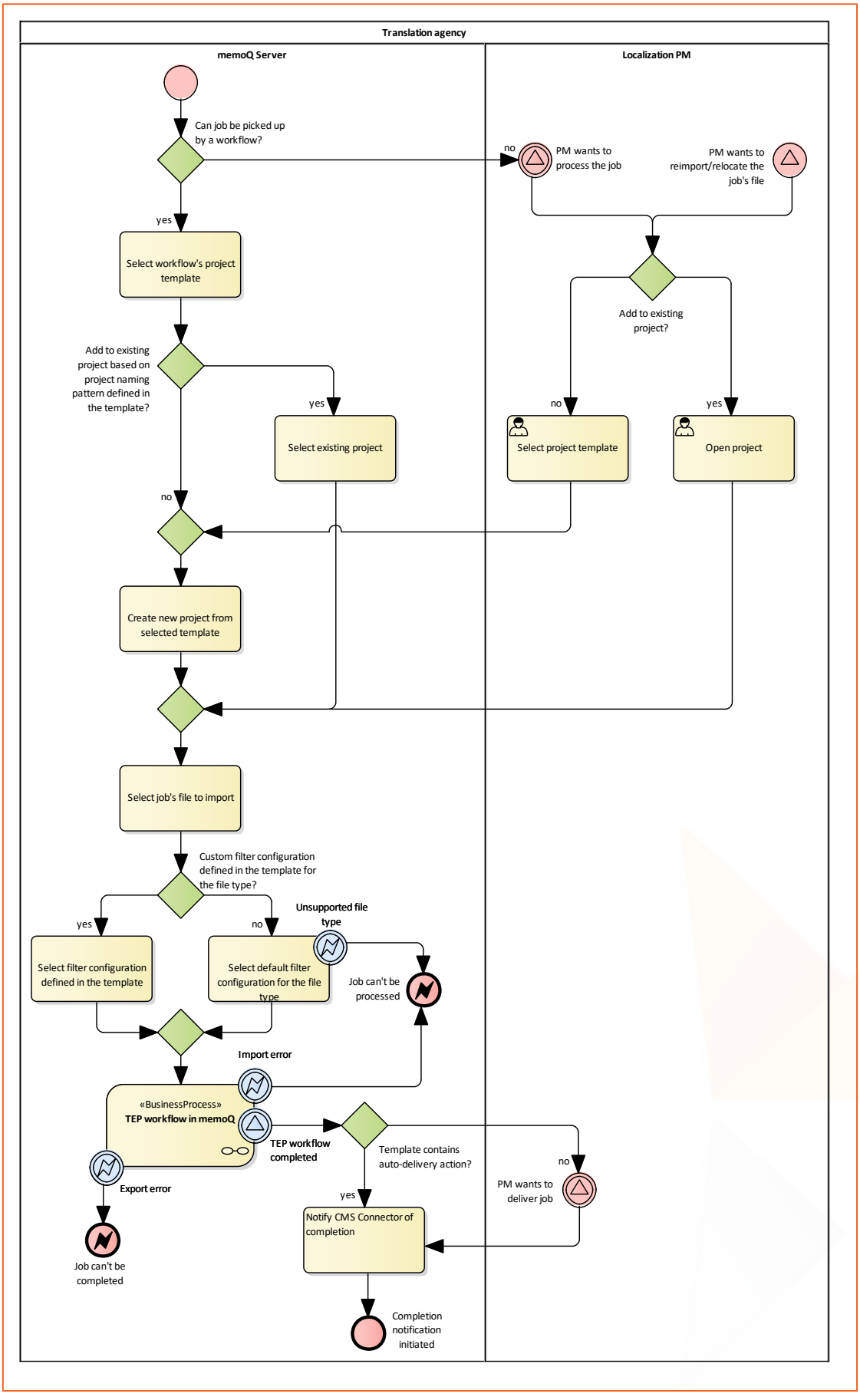
these will likely both called order. The connector then needs to upload jobs one by one, and when all jobs are uploaded, it shall announce this fact to memoQ so that it knows it does not need to wait for more jobs and can start processing.

- After an order is received, memoQ processes it. Processing includes the in-memoQ translation workflow as well. The processing that takes place in memoQ is detailed below in Translation in memoQ.
- Immediately or some time after jobs are completed, events can trigger memoQ to notify the CMS connector that some particular jobs are ready for delivery. It's up to the project template and the localization PM working in memoQ when such events occur—it may occur immediately when a job is done, or only when all jobs of an order or a memoQ project are complete. memoQ sends separate notifications about each completed job.
  - Deliver to CMS actions can be added to various events. In an improperly configured template the action may be added to an event occurring before the translation is ready (such as, to the *After document import* event). This might result in unexpected deliverables.
- The CMS Connector needs to download the jobs one by one by sending the appropriate delivery request to memoQ.
- memoQ PMs can cancel jobs. This is detailed in section Cancel Job.
  - CMS Connectors have no option to cancel submitted orders and jobs. This is a business-level limitation to avoid jobs disappearing from the CMS Dashboard and documents from memoQ projects. If a job needs to be cancelled by the end-user, they need to communicate this request to memoQ PM outside this integration.

2.3.3/A TRANSLATION IN MEMOQ

The in-memoQ translation workflow involves the processing of orders. Processing starts when an order is fully submitted. Jobs are processed one by one, along the workflow shown on the following diagram.

Figure 8. Workflow of translation in memoQ



PROCESS OVERVIEW

1. memoQ checks if there is a workflow configured for the connection with the language pair of the job.
    1.1. If there's such a workflow, it picks up the job. See what happens in section Automated Job Processing and File Import below.
    1.2. If there's no such workflow, nothing happens until a memoQ PM decides to add the job to a project. See what happens in section Manual Job Processing and File Import below.
2. The file imported from the job will go through the workflow defined in the project template and eventually controlled by the PM.
    2.1. If an event occurs for which a Deliver to CMS automated action is defined in the project template, **automated delivery** will take place:
        2.1.1. If the automated action is added to a project-level event, such as *All documents of a language complete workflow*, memoQ will notify the CMS Connector that the jobs processed in the project are all ready for delivery. The CMS Connector can download the jobs one by one.
        2.1.2. If the automated action is added to a document-level event, such as *Translator delivers document*, memoQ will notify the CMS Connector that the job containing the file triggering the event is ready for delivery. The CMS Connector can then download the job.
3. After the process takes place and the job was not delivered, it will show up in the CMS Dashboard as Completed and wait for **manual delivery**.
4. The PM can initiate manual delivery of the job. When doing so, memoQ will notify the CMS Connector that the job is ready, and the CMS Connector can download the translation.

ADDITIONAL KEY NOTES

- PMs can cancel jobs at any time before delivery (as shown in the diagram in section Translation via memoQ).
- PMs can reimport jobs. Reimporting to the same project is virtually equivalent to reimporting a file in memoQ, except for that in this case the PM cannot select a different filter configuration than that defined by the template.
    - If a different filter configuration is required, the Localization PM needs to edit the corresponding template, or the filter configuration defined in that template before reimporting.
    - Reimporting to a different project will move the file to that project and remove it from the current one.
- PMs can deliver a job multiple times. This is a means of remedy in case of, for example, communication failures and defective translations. The CMS Connector shall download such translations as many times as they are reported as ready, since the contents might have been changed—PMs are allowed to import delivered jobs to memoQ projects. This lets them fix issues found only after a previous delivery.

AUTOMATED JOB PROCESSING AND FILE IMPORT

The following happens if a job can be picked up by a workflow:

1. memoQ checks if a new project is to be created or the job shall be added to an existing one. For that it takes the project naming template of the project template set for the workflow, and parses the template.
    1.1. If no project with the parsed name exists, it will create one, based on the project template
    1.2. Otherwise it will add the job to the project with that name.
2. memoQ then checks the project's template for filter configurations.
    2.1. If a custom filter or filter configuration is set in the template for the file type of the file contained in the job, it will import the file with that filter configuration.
    2.2. Otherwise it will try to find a filter based on the file type, and apply that filter's default configuration.
        2.2.1. If no such filter exists (unknown filetype), an error is logged, and the job is not processed. At this point further processing of the job is blocked until the memoQ PM creates a template which defines a filter and a configuration for that file type.
3. Now the file is in a memoQ project, where it will go through its workflow defined in the template and eventually controlled by the PM. Job Delivery actions may trigger memoQ to announce the completion of jobs.

MANUAL JOB PROCESSING AND FILE IMPORT

The following happens if a job cannot be picked up by a workflow, or if the PM wants to import the job's file manually to a memoQ project to process it differently, or to move it to another project (relocation):

1. The PM decides if they want to add the job to an existing project or to a new one.
    1.1. If they want to add it to a new one, they select the file and create a project based on a project template of their choice.
    1.2. If they want to add it to an existing one, they open the respective project and select the corresponding command to add new jobs to that project.
2. memoQ then checks the project's template for filter configurations.
    2.1. If a custom filter or filter configuration is set in the template for the file type of the file contained in the job, it will import the file with that filter configuration.
    2.2. Otherwise it will try to find a filter based on the file type, and apply that filter's default configuration.
        2.2.1. If no such filter exists (unknown filetype), an error is logged, and the job is not processed. At this point further processing of the job is blocked until the memoQ PM creates a template which defines a filter and a configuration for that file type.

3. Now the file is in a memoQ project, where it will go through its workflow defined in the template and eventually controlled by the PM. Job Delivery actions may trigger memoQ to announce the completion of jobs.

REIMPORT AND RELOCATION

Localization PMs have the option to reimport jobs to memoQ projects. This lets them relocate the jobs to different projects, or to reimport the translatable files after they fixed an issue in the filter configuration previously used to import the file.

- **If the Localization PM wants to reimport a job to the same project:** After he template or the filter configuration is fixed, open the project, and click Add CMS Jobs on the ribbon, and select the job to reimport.
- **If the Localization PM wants to relocate the job to a new project:** The PM needs to open the CMS Dashboard in a memoQ PM client connected to the appropriate memoQ Server instance, select the job and click Import. Then the above process executes.
- **If the Localization PM wants to relocate the job to a different existing project:** Open the project to which you want to relocate the file, click Add CMS Jobs on the ribbon, and select the job to reimport.

JOB DELIVERY

Job delivery can be automated or manual:

- **Automated job delivery.** If you want memoQ to automatically announce the completion of one or more jobs so that the CMS Connector can download the translations and transfer the translated content to the third-party system, you have to add a Deliver to CMS action to the appropriate event in a project template. You can add this action to many events, however not all may make sense. Make sure you add the action to events which only occur once the workflow is complete. For example, if you want to run a review session on translations, do not add the action to the Translator delivers document event, but to the Reviewer 1 delivers document event or the Reviewer 1 delivers document event, whichever fits your requirements.
  - You can add the action to project-level events such as All documents delivered by translator or All documents of a language complete workflow. In this case memoQ will announce the completion of each job defined by the event (that is, when the all documents of a memoQ project are delivered by translator, or when all documents of a particular target language are delivered in the examples). Jobs completed earlier will wait until the criteria is met.
  - You can add the action to document-level events, such as Translator delivers document. In this case memoQ will announce completion as soon as the particular document satisfies the criteria and the event is triggered.
- **Manual job delivery.** Localization PMs can request memoQ to announce completion for completed jobs, jobs in delivery and delivered jobs any time they want, whether or not the project template for the project containing the specific job has any Deliver to CMS action or not. Doing so, memoQ immediately notifies the CMS Connector that the particular jobs are ready for delivery.

Which method the translation agency selects depends on their workflows. If they want to deliver jobs in the same sets in which they received them, that is, they want to deliver all jobs of an order at once, they need to create a workflow for all supported languages, and associate a project template, which incorporates the batch name (order name) in its project naming pattern to make sure each batch (order) goes into a separate project, and shall add a Deliver to CMS action to an event triggered when all documents are completed.

2.3.3/B CANCEL JOB

Localization PMs working in memoQ can cancel all non-delivered jobs. (To make delivered jobs to disappear from memoQ, they can delete them.) A cancelled job cannot be processed again, and the only operation allowed on cancelled jobs is to delete them.

Deleting jobs deletes all the translation files received and translated files created (whether delivered or not) in the course of processing those jobs, however the translation files imported into a memoQ projects and their translations existing there are not deleted from memoQ, having a chance to recover from inadvertent deletions—they can be deleted by deleting the respective projects, and to store them for later re-use.

memoQ Server notifies the CMS Connector about job cancellations, in a separate message for each such jobs. The CMS Connector shall not expect job completion notification for cancelled jobs, and if they try to download the results or get status information on such jobs, an error message will be returned stating the job is unknown.

2.3.3/C DELETE JOB

PMs can delete delivered and cancelled jobs. Since CMS Connector shall already have been notified of cancellation, or it must have downloaded the translation by this moment in time, no notification to CMS Connector shall be sent on deletion.

2.3.4. RECAP

In this section we've discussed the high-level workflows related to memoQ's CMS integration feature. The next section, as a kind of logical design, presents the communication protocol of the interoperation.

3. COLLABORATION PROTOCOLS

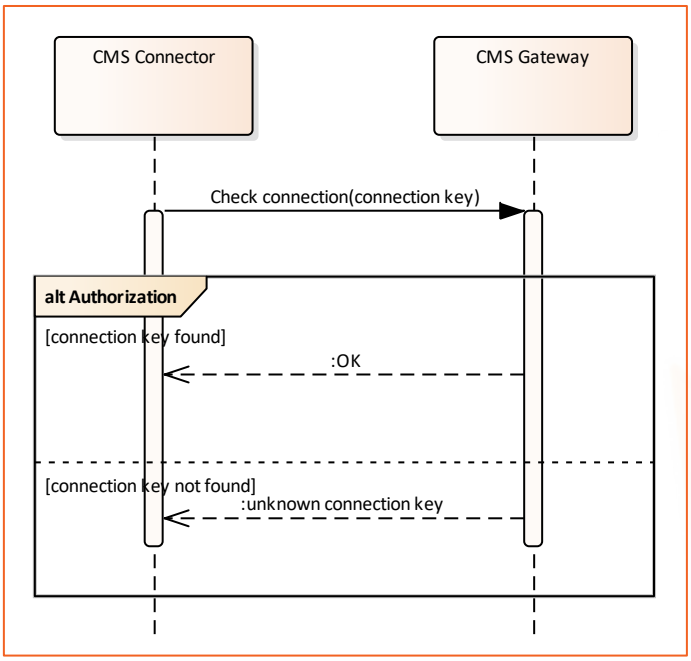
This chapter defines the sequence of messages to exchange in order to carry out a workflow step at logical level. The operations contain UML sequence diagrams with logical-level definition of operations. Following each diagram, a table explains the messages and maps the logical operation to actual requests by referring to the appropriate section of implementation-level design (chapter Implementation – API Description). This section helps you to understand the logic of the collaboration, while the implementation-level design details the requests to make at code level.

3.1. CONNECTION REGISTRATION

The registration-related operations only contain one automated collaboration, detailed in the next section.

3.1.1. CHECK CONNECTION HEALTH

Figure 9. Collaboration protocol of performing connection health check



CHECK CONNECTION	
Key parameters	connection key
Description	The purpose of this message is to check if the connection can be established with memoQ.
Notes on the response	The response code informs the caller if the client could be authenticated on the server and the connection is approved. An appropriate error is returned for rejected connections so that the caller knows the request reached the server, only the connection key

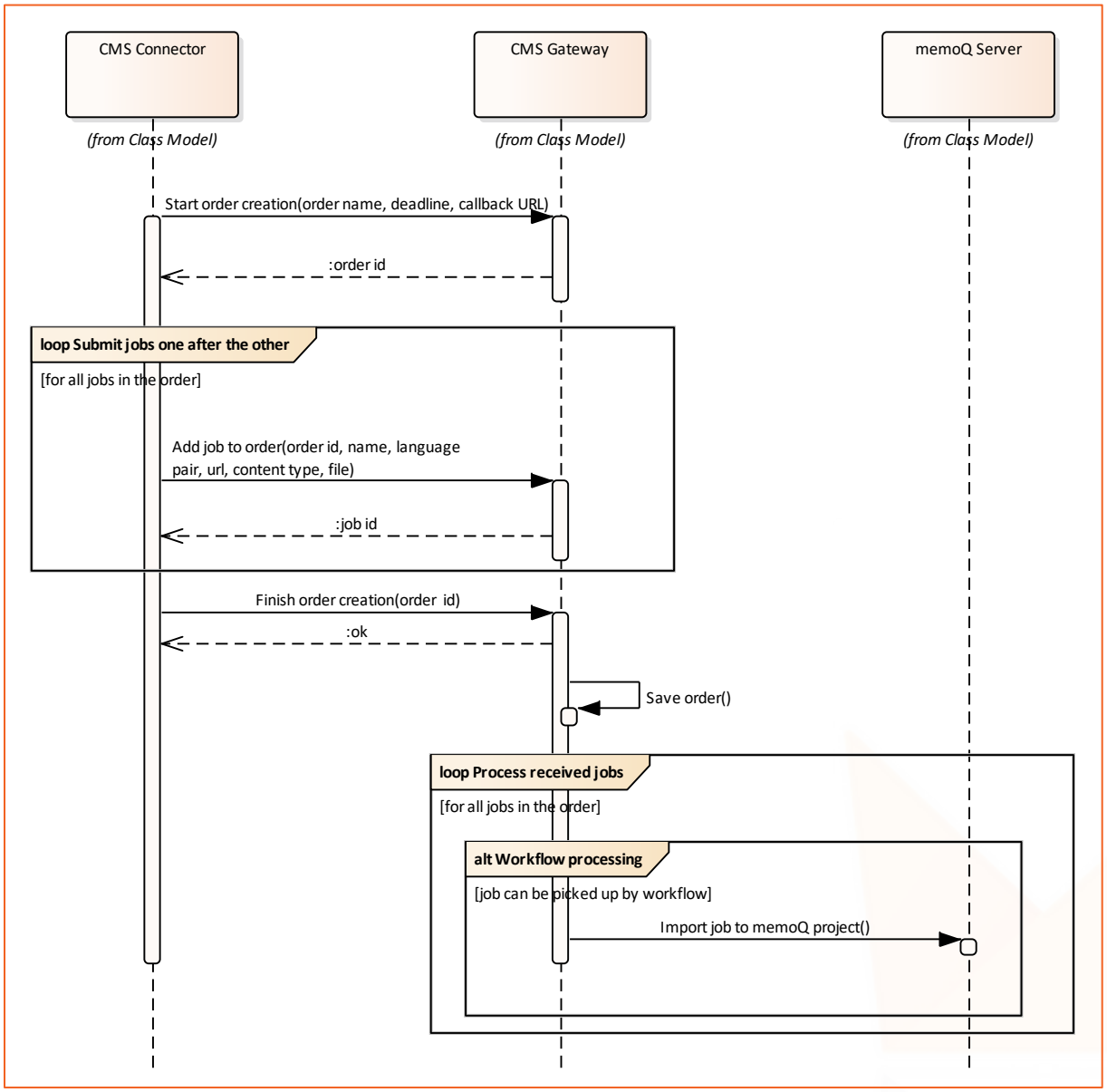
	was wrong. You can use the returned info to see if it matches with those registered for this connection in the CMS Connector.
Implementation	The operation is implemented by the /client GET request. See Client information. EVOLUTION NOTE: In the future, a specific register request may be added to the CMS API.

3.2. PROJECT AND JOB MANAGEMENT

This section contains the collaborations for creating projects, getting status information and downloading deliverables.

3.2.1. SUBMIT ORDERS AND JOBS

Figure 10. Collaboration protocol of submitting orders



Order submission consists of announcing the beginning of order creation, submitting the jobs making up the order, and announcing the end of job submission. The operations are detailed in the next sections.

START ORDER CREATION

Key parameters	order name, deadline, callback URL

Description	The CMS Connector first needs to announce it wants to submit an order. memoQ Server prepares to receive new jobs. memoQ will use the provided callback URL for announcements coupled with job state changes. Note that memoQ currently ignores the deadline.
Notes on the response	memoQ returns the order ID required for acting upon the order later
Implementation	The operation is implemented by the /projects POST request. See Create Order. EVOLUTION NOTE: In the documentation and the source code of the API, order is called project. This is planned to be replaced with order.

ADD JOB TO ORDER

Key parameters	order id, name, language pair, url, content type, file
Description	This operation adds a job to the order being submitted. The job is added to the specified order. The job definition includes the gzipped translatable file as an attachment, the type of the file (in the form of file extensions), the source and target language, and the URL on which the source is available (for previewing during translation). This operation needs to be invoked for each job in the order's intended scope.
Notes on the response	memoQ returns the job ID required for acting upon the job later. If the order specified does not exist or is closed, error is returned.
Implementation	The operation is implemented by the /projects/{projectId}/jobs POST request. See Submit Job. EVOLUTION NOTE: In the documentation and the source code of the API, order is called project. This is planned to be replaced with order.

FINISH ORDER CREATION

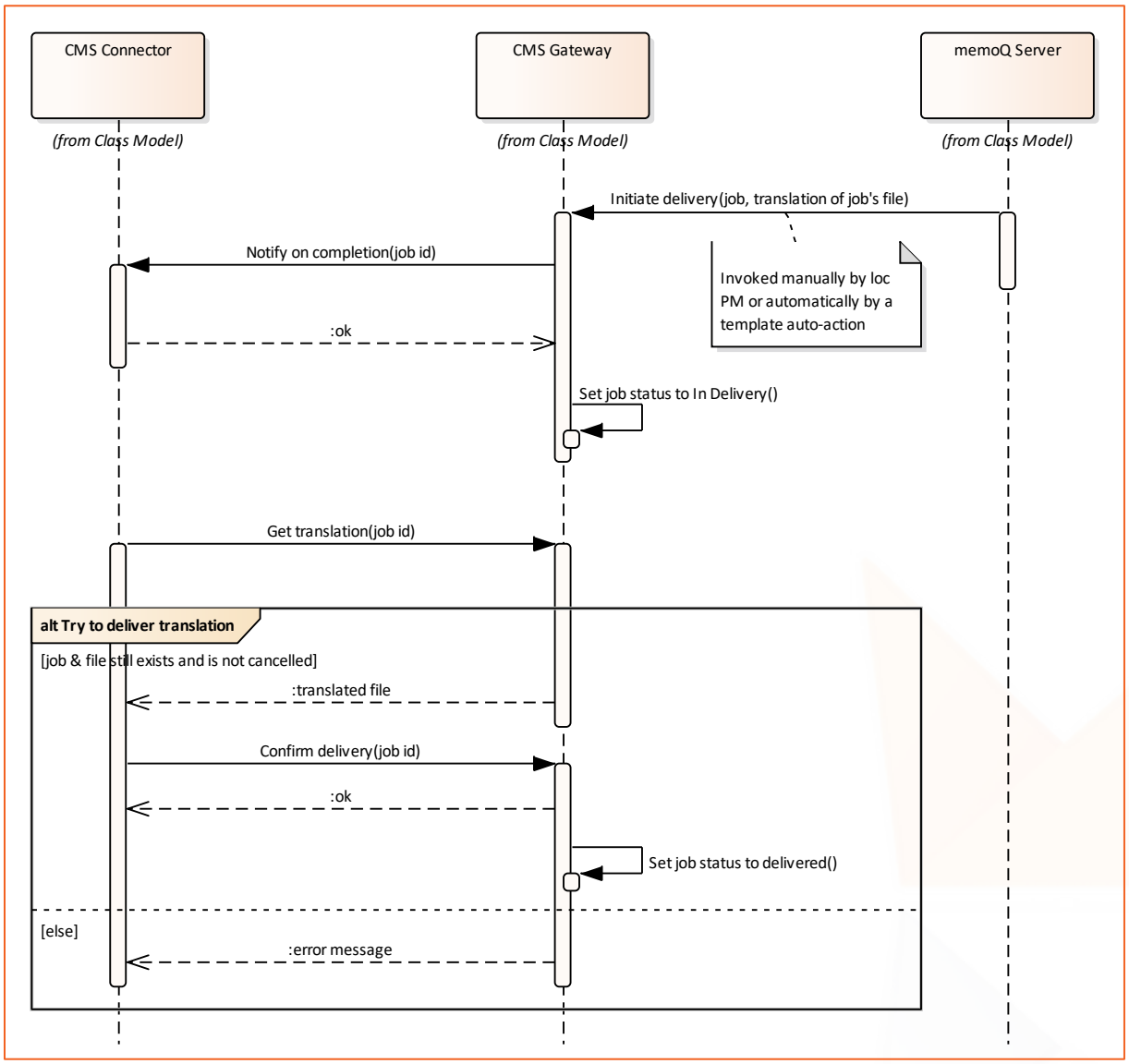
Key parameters	order id
Description	Once all jobs are submitted, the CMS Connector needs to let know memoQ Server about this by closing the order creation.
Notes on the response	memoQ lets the CMS Connector know if the operation succeeded, or what error occurred.

Implementation	The operation is implemented by the /projects/{projectId} PATCH request. See Update Order status. EVOLUTION NOTE: In the documentation and the source code of the API, order is called project. This is planned to be replaced with order.

3.2.2. DELIVER JOBS

As said in Translation via memoQ, delivering jobs takes two steps: the CMS Gateway notifies the CMS Connector via the CMS API that a job is completed and is available for delivery. In response, the CMS Connector can download the translation.

Figure 11. Collaboration protocol of delivering jobs



NOTIFY ON COMPLETION

Key parameters	job id

Description	The notification is sent to the callback URL specified for the container order. The CMS Connector shall acknowledge the reception of the notification. If you prefer a pull model and don't want to use notifications because, for example, you don't want to make CMS Connector publicly available, and therefore you don't specify a real callback URL, the CMS Connector won't receive such notifications. The jobs in this case may appear as In Delivery or Delivery Error in memoQ until the CMS Connector downloads them.
Notes on the response	The CMS Connector needs to confirm the acknowledgement of this completion event. If the job cannot be found, it shall return an appropriate error.
Implementation	The operation is implemented by the /notification **POST** request, submitted by memoQ Server, and the TranslationReady status submitted in the request means the notification is a completion notification. See Notification on Job Status Change.

GET TRANSLATION

Key parameters	job id
Description	The CMS Connector can download the translation of the job specified in the job id parameter using this request.
Notes on the response	The response contains the translated file in gzipped format.
Implementation	The operation is implemented by the /jobs/{translationJobId}/translation **GET** request. See Get Translation.

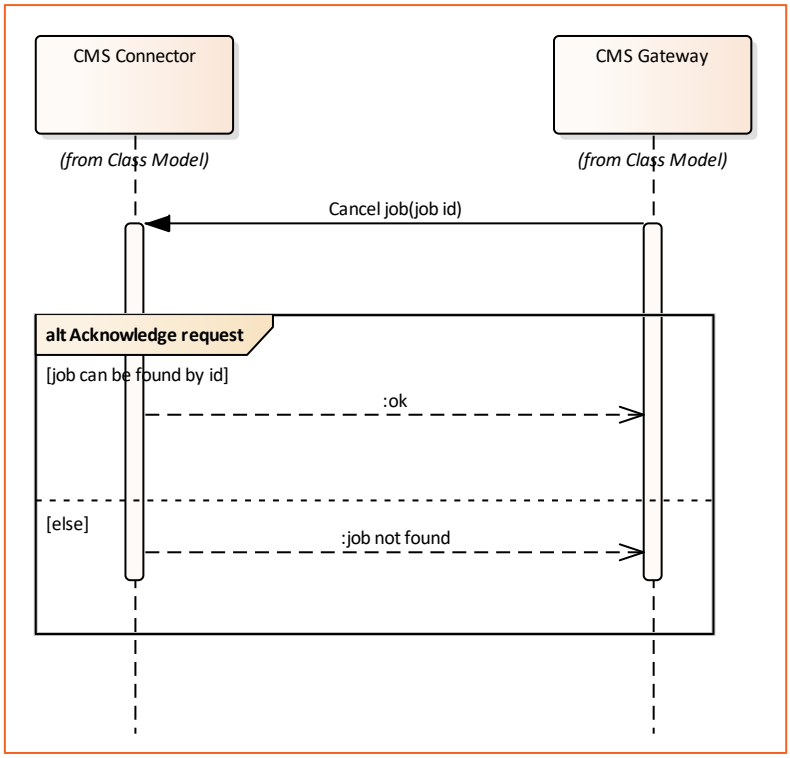
CONFIRM DELIVERY

Key parameters	job id
Description	The CMS Connector shall inform memoQ Server about the successful download so that it can mark the job as Delivered in memoQ.
Notes on the response	Just an acknowledgement of the reception of the confirmation.
Implementation	The operation is implemented by the /jobs/{translationJobId}/ **PATCH** request, with the NewStatus parameter set to DeliveredToSource. See Update Job Status.

3.2.3. CANCEL JOBS

Remember, only memoQ Server can cancel CMS jobs, the CMS Connector cannot. The idea behind is that the translation process might have been started by the translation agency, and cancelling a job would pose the risk that the agency works on cancelled tasks. If the customer wants to cancel a job, they need to contact the localization TM, negotiate the details, and the localization PM will cancel the job.

Figure 12. Collaboration protocol of cancelling a job by memoQ Server



NOTIFY ON COMPLETION

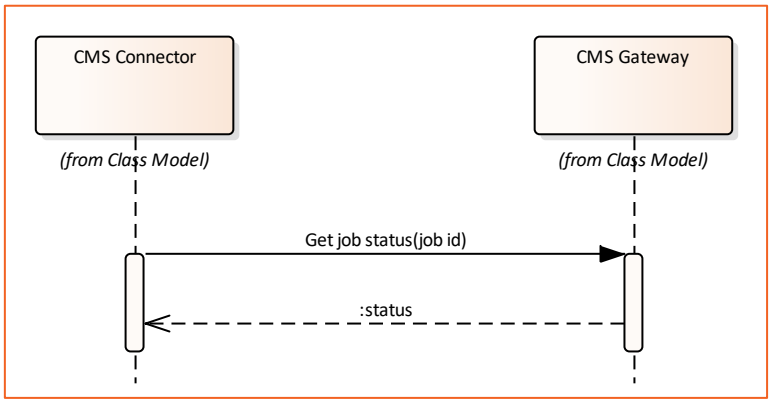
Key parameters	job id
Description	The notification is sent to the callback URL specified for the container order. The CMS Connector shall acknowledge the reception of the notification. If you prefer a pull model and don't want to use notifications because, for example, you don't want to make CMS Connector publicly available, and therefore you don't specify a real callback URL, the CMS Connector won't receive such notifications. The jobs will remain cancelled in memoQ.
Notes on the response	The CMS Connector needs to confirm the acknowledgement of this cancellation. If the job cannot be found, it shall return an appropriate error.
Implementation	The operation is implemented by the /notification POST request, submitted by memoQ Server, and the Cancelled status submitted in the request means the notification is a completion notification. See Notification on Job Status Change.

3.3. MISCELLANEOUS

3.3.1. GET JOB STATUS INFORMATION

Once the CMS Connector successfully submitted an order, it can query the status of the contained jobs. You can use the status to route your logic—for example you can try to download a job if it's complete, if you prefer a pull model for delivery instead of the push model in which memoQ notifies you of job completion.

Figure 13. Collaboration protocol of getting CMS job status information



GET JOB STATUS	
Key parameters	job id
Description	Submit a request for the status of a particular job.
Notes on the response	memoQ returns all information about the job, including its status.
Implementation	The operation is implemented by the /jobs/{translationJobId} GET request. See Update Order status.

3.3.2. GET CLIENT INFORMATION

A metadata of CMS Connections in memoQ is the client. A client is a record of an ID, a name and an email address, and has currently no purpose in memoQ, it's just descriptive data. You can use client information to map orders and jobs to clients.

WHAT IS CLIENT INFORMATION GOOD FOR?

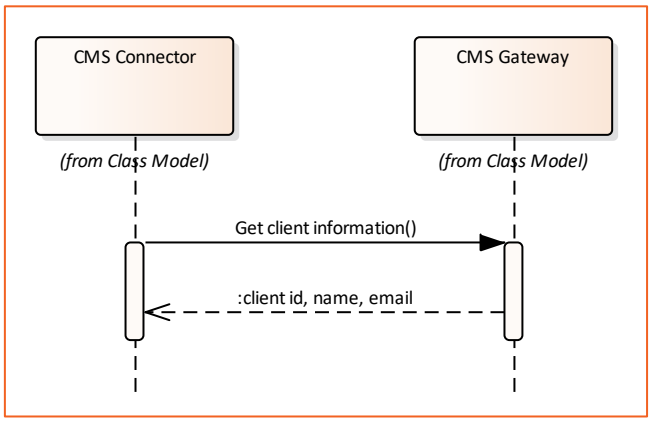
Client information was implemented to conform to the business model of our first CMS integration partner, OnTheGoSystems (OTGS), the vendor of the leading WordPress localization platform, WPML. OTGS uses client id in WPML this way:

- Translation agencies can use WPML's integration services for free, but they can opt-in to an affiliate program. In exchange for getting recommended by WPML in WPML's pages, increasing the chance that WPML translation buyers contact them, participating agencies pay a (small) fee for the number of words transmitted to them for translation. OTGS wanted to use weighted wordcounts to avoid double charging for what is translated only once.
- OTGS creates a separate TM for each client within the realm of each WPML customer. That is, if a translation agency works for 2 customers, and the first customer has 3 websites, while the second has only 1, then two TMs are created.
- Contents of CMS jobs submitted from any of the 3 websites of the first client are loaded into the first TM, and the contents of translations jobs submitted from the website of the second client are loaded into the second TM.
- Before adding the contents to the TM, WPML performs an analysis on the contents to be added to the TM and applies a weighted cost model to calculate effective word count.
- This way translation agencies need to pay after the weighted word count at client level. If the same segment is submitted from website 1 and 3 of the first client in this example, only the first occurrence will increase the fee paid to WPML, as they likely can charge it only once to the client.
  - If the same segment (such as "All rights reserved.") is submitted from the website of the second client, the full word count will contribute to the fee calculation.
  - This also means that if a job needs to be submitted the second time for an error, the agency does not need to pay double price.

It's up to you whether you want to use a consumption measure model like this or not. memoQ supports this by making client a mandatory property of connections, including a client ID unique on memoQ Server instance-level, client name and email, which you can use for communication.

Customers can enter client IDs manually, so if the CMS Connector uses client IDs, the translation agencies may be requested to use those IDs, and multiple instances of memoQ Servers can use the same client ID as well.

Figure 14. Collaboration protocol of getting client information



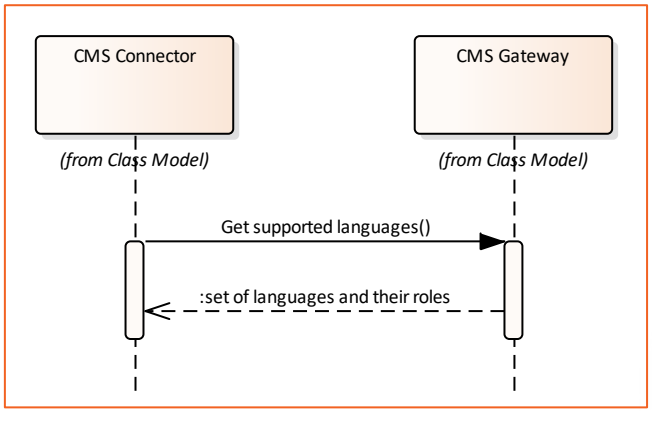
GET CLIENT INFORMATION	
Key parameters	*none*
Description	Request for client information
Notes on the response	memoQ returns the unique ID, the name and the contact email address of the client associated with the CMS Connection used to communicate between the caller instance of the CMS Connector and the called instance of memoQ Server.
Implementation	The operation is implemented by the **/client** **GET** request. See Client information.

3.3.3. GET SUPPORTED LANGUAGES

CMS Connectors can query the set of supported languages by memoQ. The response contains the list of three-letter language codes in the ISO format. The CMS Connector needs to specify these language codes in jobs, or memoQ Server won't be able to process the jobs. If a language is marked as only-target, it cannot be the source language of jobs, or memoQ Server will fail to process the jobs.

The purpose of getting language codes is, therefore, to speak the same language and provide compatibility between the integrated system. If CMS Connector knows these codes, the operation does not need to be performed, but be aware that the set of supported languages may still increase in memoQ time to time.

Figure 15. Collaboration protocol of getting list of supported languages



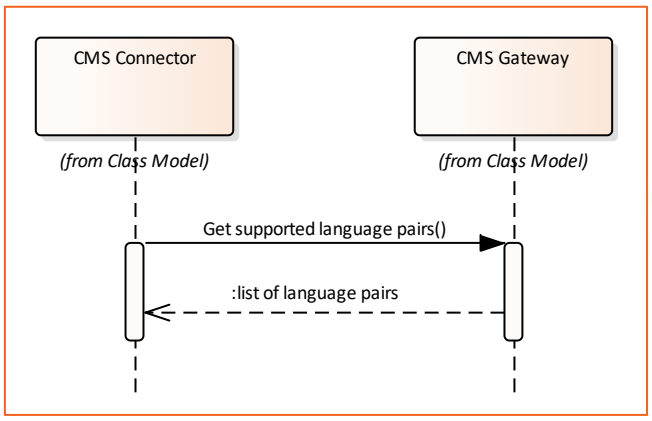
Note: If you want to know the language pairs supported by a specific CMS Connection, look for Get Language Pairs Supported by the Connection below.

GET SUPPORTED LANGUAGES	
Key parameters	*none*
Description	The operation gets the list of language codes supported by memoQ Server for the CMS Connector to use them in jobs.
Notes on the response	A list of ISO 3-letter language codes and the role (source and target, or target only) of each language
Implementation	The operation is implemented by the /languages GET request. See Get Language Codes.

3.3.4. GET LANGUAGE PAIRS SUPPORTED BY THE CONNECTION

The administrators of memoQ Server needs to specify the source and target languages supported when creating a CMS connection (and they can later edit this information). The CMS Gateway will only accept jobs in any of the possible language pairs based on the supported source and target languages. You can get this list with this operation. If you submit a job in a language pair not present in the list, the CMS Gateway will deny accepting it.

Figure 16. Collaboration protocol of getting the list of language pairs supported by the CMS connection



GET SUPPORTED LANGUAGES	
Key parameters	*none*
Description	The operation gets the list of language pairs enabled on the CMS Connector's CMS connection in memoQ Server.
Notes on the response	A list of ISO 3-letter language code pairs.
Implementation	The operation is implemented by the /languagePairs GET request. See Get Supported Language Pairs.

4. API REFERENCE

This chapter serves as a reference for the methods provided by the CMS API. The scenarios in which one or the other method can be used is described in the previous chapter.

We've created a reference implementation of CMS Connectors. Read more in chapter 5, Sample CMS Connector.

SWAGGER: EXPLORE IMPLEMENTATION-LEVEL DOCUMENTATION

The implementation-level documentation is incorporated in the API's source code, and can be exposed using a tool called Swagger. It exposes a browser-based user interface which lets you know consult the parameters and return values of requests along with their documentation, and also to try out the calls yourself.

Swagger can be accessed at http://localhost:8080/memoqservercmsgateway/v1/swagger/ on the computer running memoQ Server. For each method below, we also disclose the direct Swagger URL displaying the details of that method. If you read this documentation on the computer running memoQ Server, just click the links, otherwise copy them to a browser running on the computer hosting memoQ Server.

CONTRADICTIONS

Should you find any information on the Swagger UI that contradicts with this document, note that the information on Swagger's UI is authoritative, since it comes from the actual source code. In such cases, please report the contradictions to memoQ Support so that we can fix the documentation. Thank you.

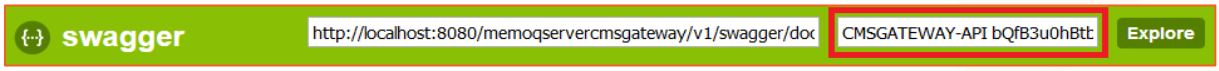
4.1. AUTHORIZATION

The caller has to send the connection key in the authorization header of each request submitted to memoQ Server. The content of the header field has to be in the following format:

```
CMSGATEWAY-API [connection key]
```

If you would like to call the API from Swagger, then you have to type the secret into the api_key field in the above mentioned format:

Figure 17. How to enter the connection key in Swagger



4.2. CONNECTION MANAGEMENT

This section lists the requests used to query connection-level information.

4.2.1. CLIENT INFORMATION

GET /client

The caller can access information about the client through the API (client id, client name, and email) under the client resource name (relative URL).

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Client/Client_GetClientId

4.2.2. GET LANGUAGE CODES

GET /languages

Language codes in all cases are three+(two) letter ISO language codes, such as `fre`, `eng`, and `eng-US`. To find out what are the supported language codes and what their mapping to other language codes are, see `LanguageAllData.xml`. The three-letter language codes conform to ISO 639, and the two-letter country codes conform to ISO 3166. Both standards have different versions (e.g.: ISO 639-1, ISO 639-2, etc.), and memoQ may use a mixture of these. You can get information about the languages supported by the memoQ Server by submitting this request.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Client/Client_GetSupportedLanguages

NOTE: All language codes listed in `LanguageAllData.xml` are supported as target languages, but not all as source languages: target-only languages have the OnyTarget field set to true. The returned list of languages is independent from the client, it includes all languages supported by the memoQ Server product.

4.2.3. GET SUPPORTED LANGUAGE PAIRS

GET /languagePairs

Language pairs are specific for CMS connections. Make sure you only submit jobs in the supported language pairs, since jobs of other language pairs won't be accepted.

The returned list is empty if the administrator of memoQ Server did not specify any source and target languages. In this case the given CMS connection supports all combination of all languages supported by memoQ Server.

The CMS Connector may provide an option to its users and/or automatism to submit this request on-demand or in response to errors to see if the set of the supported language pairs have been changed in memoQ Server.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Client/Client_GetConnectionLanguagePairs

4.3. ORDER MANAGEMENT

This section lists the requests related to managing orders (remember, orders are currently called projects in the API).

4.3.1. CREATE ORDER

POST /projects

Creates a new order based on the request parameters and returns the created order in the response. The response's location header contains the URL of the order created.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Project/Project_CreateProject

4.3.2. UPDATE ORDER STATUS

PATCH /projects/{projectID}

Updates the status of an order based on the request parameter. The single possible status for now is Committed, which tells the CMS Gateway all jobs of the order has been submitted.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Project/Project_UpdateProjectStatus

4.3.3. GET ORDER INFORMATION

GET /projects/{projectID}

Gets information about an order specified in the request parameter.

Details http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Project/Project_GetProject

4.3.4. LIST ORDERS

GET /projects

Lists all the orders of this CMS connection existing on CMS Gateway.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Project/Project_GetProjects

4.4. JOB MANAGEMENT

This section lists the requests available to act upon CMS jobs.

4.4.1. SUBMIT JOB

POST /projects/{projectId}/jobs

Submits a new CMS job for an order based on the request parameter, and returns the created job in the response. The request's content type has to be `multipart/form-data`. The first part has to contain information about the new job, and its name has to be translationJob. The second part of the request has to contain the file to be translated and its name has to be file. The file is recommended to be gzipped, and in this case the MIME type shall be `application/x-gzip`. The response's location header contains the URL of the job created.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Project/Project_CreateTranslationJob

4.4.2. UPDATE JOB STATUS

PATCH /jobs/{translationJobId}

Updates the status of a translation job based on the request parameter. This is currently used to let memoQ Server know the CMS Connector managed to download the translation of a completed job.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Job/Job_UpdateTranslationJobStatus

4.4.3. GET JOB INFORMATION

GET /jobs/{translationJobId}

Gets information about a translation job based on the request parameter. You can use this request to check and match the job details stored in memoQ and in the CMS Connector, and to get progress information on job processing.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Job/Job_GetTranslationJob

4.4.4. GET ORDER SCOPE

GET /projects/{projectId}/jobs

Lists the translation jobs of an order based on the request parameter. You can use this request to check if all jobs you wanted to add to the order are actually part of the order. Deleted jobs are not part of the list returned.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Project/Project_GetTranslationJobs

4.4.5. GET TRANSLATION

GET /jobs/{translationJobId}/translation

This request can be used to download translation of a job based on the request parameter. If the job is not complete, an error will be returned. Otherwise, the response contains the translated file itself if gzipped format.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Job/Job_GetTranslation

4.5. NOTIFICATIONS

4.5.1. NOTIFICATION ON JOB STATUS CHANGE

POST /notification

The CMS Gateway sends a notification if the status of a job becomes complete (`TranslationReady` at code level) or cancelled (`Cancelled` at code level) in the CMS gateway. The CMS gateway sends the details in the POST request body to the callback URL provided during order creation.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Notification/Notification_TranslationJobNotificationCallback

4.5.2. TEST NOTIFICATION

POST /test/jobs

There is an additional notification for testing purposes that also contains the specification of the notification object. You can trigger the mentioned notification from the CMS Gateway Mockup Server application, if you send a request to this test endpoint. In addition, notifications will be triggered during normal operation on both the mocked and the actual gateway implementations.

Details: http://localhost:8080/memoqservercmsgateway/v1/swagger/ui/index#!/Test/Test_TriggerTranslationJobNotification.

All notifications are sent to the same shared callback URL as a JSON object. Different notification types are distinguished by the Type property and all notification-specific data is sent in the Payload property. An example for the complete JSON response:

```
{
  "Type": "TranslationJobStatusChanged",
  "Payload": {
    "TranslationJobId": 0,
    "NewStatus": "Delivered"
  }
}
```

4.6. ERROR HANDLING

The Swagger documentation contains the possible HTTP status codes for each requests. If any problem occurs during processing a request, the CMS Gateway returns a JSON object describing the problem. This allows to distinguish different issues under the same HTTP status code. The returned object has the following members:

- ErrorCode: The code uniquely identifies the problem.
- Message: The detailed description of the problem.

The table below contains the possible http status codes, error codes and messages.

STATUS CODE	ERROR CODE	MESSAGE
UNAUTHORIZED (401)	AuthenticationFailed	Authentication failed, please check provided secret.
FORBIDDEN (403)	ProjectDoesNotBelongToTheUser	Project does not belong to the user which was found based on the provided secret.
NOTFOUND (404)	ProjectDoesNotExist	Project does not exist.
CONFLICT (409)	ProjectIsAlreadyCommitted	The project has already been committed.

STATUS CODE	ERROR CODE	MESSAGE
CONFLICT (409)	ProjectDoesNotContainAnyJobs	Project does not contain any jobs.
CONFLICT (409)	TranslationJobSourceLangIsInvalid	The source language code is invalid.
CONFLICT (409)	TranslationJobTargetLangIsInvalid	The target language code is invalid.
CONFLICT (409)	TranslationJobSourceLangIsTargetOnly	The source language is target only language.
CONFLICT (409)	TranslationJobLangPairIsNotSupported	The language pair is not supported by the connection.
FORBIDDEN (403)	TranslationJobDoesNotBelongToTheUser	Translation job does not belong to the user which was found based on the provided secret.
NOTFOUND (404)	TranslationJobDoesNotExist	Translation job does not exist.
CONFLICT (409)	TranslationJobHasHasAlreadyBeenCancelled	Translation job has already been cancelled.
CONFLICT (409)	TranslationJobHasHasAlreadyBeenMarkedDelivered	Translation job has already been marked as delivered.
CONFLICT (409)	TranslationJobIsNotReady	Translation job is not ready.
BADREQUEST (400)	InvalidArgument	[Information about the invalid arguments]
BADREQUEST (400)	MediaTypeIsNotSupported	Only the media type "multipart/form-data" is supported.
BADREQUEST (400)	TranslationJobPartIsMissingFromRequest	The part "translationJob" is missing from the request.
BADREQUEST (400)	FilePartIsMissingFromRequest	The part "file" is missing from the request.

STATUS CODE	ERROR CODE	MESSAGE
INTERNALSERVER-ERROR (500)	InternalServerError	An internal error occurred on the server. Check the log for details.

5. SAMPLE CMS CONNECTOR APPLICATION

We've created a sample implementation of the CMS Connector, Sample CMS Connector or SCC, available for download on memoQ's website. You can use this sample implementation to play with the CMS API, and you are free to change the source code as required.

The sample client application can be used to connect JIRA to memoQ Server, or to connect a local file folder to memoQ Server:

- In the Jira mode, the sample client is connected to a Jira project. It picks up issues you define in a Jira filter, and submits them to memoQ Server. Then the issues enter In progress state (or what you define), until the respective jobs are not delivered, and then finally they enter Done state (or what you define). This mode can be used to test integration lifecycle and the API with state-aware systems.
- In the local folder mode, the sample client picks up all files found in the specified directory, and submits them to memoQ Server, using their filename extension as the file type. The translated files are delivered to a subfolder. This mode can be used to see if a files of a specific type or with specific contents can be properly processed in memoQ.

The application, written in C#, is distributed in source code form, and it is a Microsoft Visual Studio project. If you just want to play with it first, you can run its executable from the \bin\Debug folder, without any installation steps.

5.1. CONFIGURING THE APPLICATION

The CMS API Sample Client application has a configuration file. You can use this to specify connection details to a Jira instance under your control used for testing purposes. You can find this file in `c:\Users\<your-user-name>\AppData\Local\memoQ Sample CMS Connector\`, and is named `memoQ Sample CMS Connector.config`. The file is created when you run the application the first time.

You can configure the settings described in the following table.

PARAMETER NAME	DESCRIPTION
cmsApiUrl	The service endpoint provided by memoQ Server. To get this, launch memoQ PM, connect to memoQ Server, open Server Administrator, click CMS Connections, and select the corresponding CMS connection, then click Copy client connection information to the Clipboard. Then paste the contents to a text editor.
cmsApiSecret	The connection key for memoQ Server

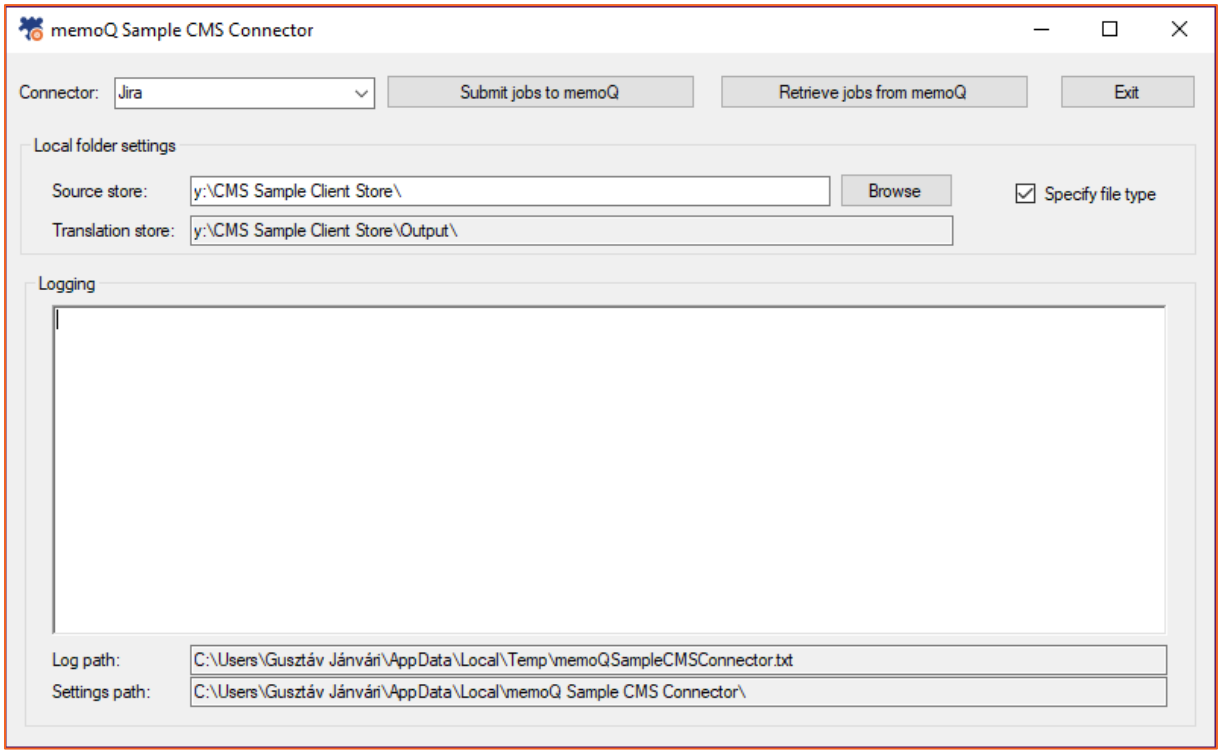
PARAMETER NAME	DESCRIPTION
jiraApiUrl	The URL to access your Jira instance, in the form of `https://<jira-URL>/rest/api/latest/`
jiraApiUser	The email address of the Jira user used to access Jira.
jiraApiPassword	The password of the user specified in the jiraApiUser parameter. Since the password is stored as clean text, it is recommended to create a dedicated users with the least required set of privileges.
jiraApiQuery	The query to run on Jira to get the issues used for your tests. The simplest way is to create a dedicated test project with the simplest workflow, and specify it in the form of Project = "<Project name>" AND status = "To Do". This means that the sample client will pick up each issues in **To Do** state and will submit them to memoQ.
xliffSource	The source language code for the XLIFF files to submit to memoQ. Use two-letter codes like **en**.
xliffTarget	The target language code for the XLIFF files to submit to memoQ. Use two-letter codes like **de**.
memoQSource	The memoQ-compatible source language code. Use three-letter codes like **eng**.
memoQTarget	The memoQ-compatible source language code. Use three-letter codes like **ger**.
loggingLevels	Leave as is
TransitionForStartingTranslation	The status to which issues should be put when they are submitted to memoQ Server. We believe 21 is a standard value for **In Progress** and doesn't need to be touched.
TransitionForFinishingTranslation	The status to which issues should be put when their translations are downloaded from memoQ Server. We believe 21 is a standard value for **Done** and doesn't need to be touched.
inputFolder	Specify the folder where you want to store the files to submit to memoQ Server in the local folder mode. The application will pick up all files from this folder only, ignoring its subfolders. Translations will be delivered to the subfolder named **Output** of this folder. You can also set this folder on the tool's UI, and it will save it to this setting, so you don't need to set it all the time you start the application.

5.2. USING THE APPLICATION

The followings briefly describe how to use the application. Feel free to consult the source code for more details.

The application displays the following window.

Figure 18. Sample CMS Connector's window



If you submit a job to memoQ in one or the other operation mode, be sure that the application is the same mode before you retrieve its translation.

5.2.1. JIRA MODE

SCC submits issues in XLIFF format, after converting the JSON received from Jira. The job payloads (the transmitted files) will contain the Description field of the tickets, and the job name will be the Issue key (issue ID). It ignores the title (Summary field)—but feel free to add it as your first modification to the sample.

Select Jira in the Connector drop-down box. Create some new issues in the connected Jira project, then click Submit jobs to memoQ. You'll see what is happening, and if the submission is successful, you'll see the new order and its jobs in the CMS Dashboard of memoQ (after you click Load/Refresh). The issues in Jira will move to In Progress.

Upon delivery, memoQ sends a notification to SCC, however you'll need to click Retrieve jobs from memoQ to request SCC to download the translated files. This lets you investigate what happens between the two events, but you can automate this download option as the second modification to the source code, if you like. The issues in Jira will move to Done, and you'll see the translation in the Description field.

5.2.2. LOCAL FOLDER MODE

Just copy a bunch of files to a folder you like. Pay attention to file extensions, since they will be used as the content type, which memoQ will use to determine which filter and configuration to use for the various files.

Select Local folder in the Connector drop-down box. Make sure Source store contains the path you need. Now simply click Submit jobs to memoQ. You'll see what is happening, and if the submission is successful, you'll see the new order and its jobs in the CMS Dashboard of memoQ (after you click Load/Refresh).

Upon delivery, memoQ sends a notification to SCC, however you'll need to click Retrieve jobs from memoQ to request SCC to download the translated files. This lets you investigate what happens between the two events, but you can automate this download option as the second modification to the source code, if you like. The received files will be saved to the path shown in the Translation store field.

The Specify file type checkbox is there only for backward-compatibility reasons, and you should not clear it. If you do so, SCC won't specify the content type for jobs, and memoQ will use the WordPress WPML filter for the files, what is likely against your intentions.

5.3. SOURCE CODE

Feel free to consult the source code. Here we just give a few hints as a head start to navigate it:

- Low-level communication, assembling requests and parsing responses is implemented in the `mQCMSAPICommInterface` and the `mQCMSAPI_Session` class.
- The class `DataStructures` defines the API request parameter structures. These will be converted to JSON upon assembling requests and parsing responses.
- The classes `TransferCMS2memoQ` and `TransfermemoQ2CMS` implement order submission and job download logic.
- `CMS_Session_Jira` takes care for Jira communication, and `CMS_Session_Virtual` handles file system operations for the local folder mode.

6. FREQUENTLY ASKED QUESTIONS

Do I need to host the CMS Connector as a publicly available service?

No, however in this case you cannot specify a callback URL for orders, meaning the CMS Connector needs to use polling technique to get completed translations. See Do I need to specify a callback URL for orders? for details.

For development and testing purposes, you can use the Sample CMS Connector on the computer running your memoQ Server sandbox instance, and therefore you can use localhost as the host name in URLs, and you can host the CMS Connector on that machine, too.

Do I need to specify a callback URL for orders?

Yes. You don't need to rely on them, however, and you may even try specifying fake URLs. In this case memoQ won't be able to use a push model to instantly notify the CMS Connector that a job has been completed. It may also cause the completed jobs to be stuck in an In Delivery state or to reach Delivery Error state.

If you don't specify a valid callback URL, the CMS Connector needs to regularly check the status of the submitted jobs, and download the completed ones. This requires a careful selection

of polling frequency, since a translation workflow may take days. If you poll the server too often, you'll generate a lot of superfluous traffic. If you choose a low frequency, completed jobs will be delivered later as they may rest on the server for hours, until the next status information request is received.

The Sample CMS Connector does not use valid callback URLs.

The CMS Connector has downloaded a translation. Then it received a completion notification for the same job again. Shall it download the translated file again?

Yes. The contents of the translation might have been changed. For example, the translation agency might have fixed an error. You should always assume a repeated delivery is for a good reason, and that the latest translated file is the proper one.