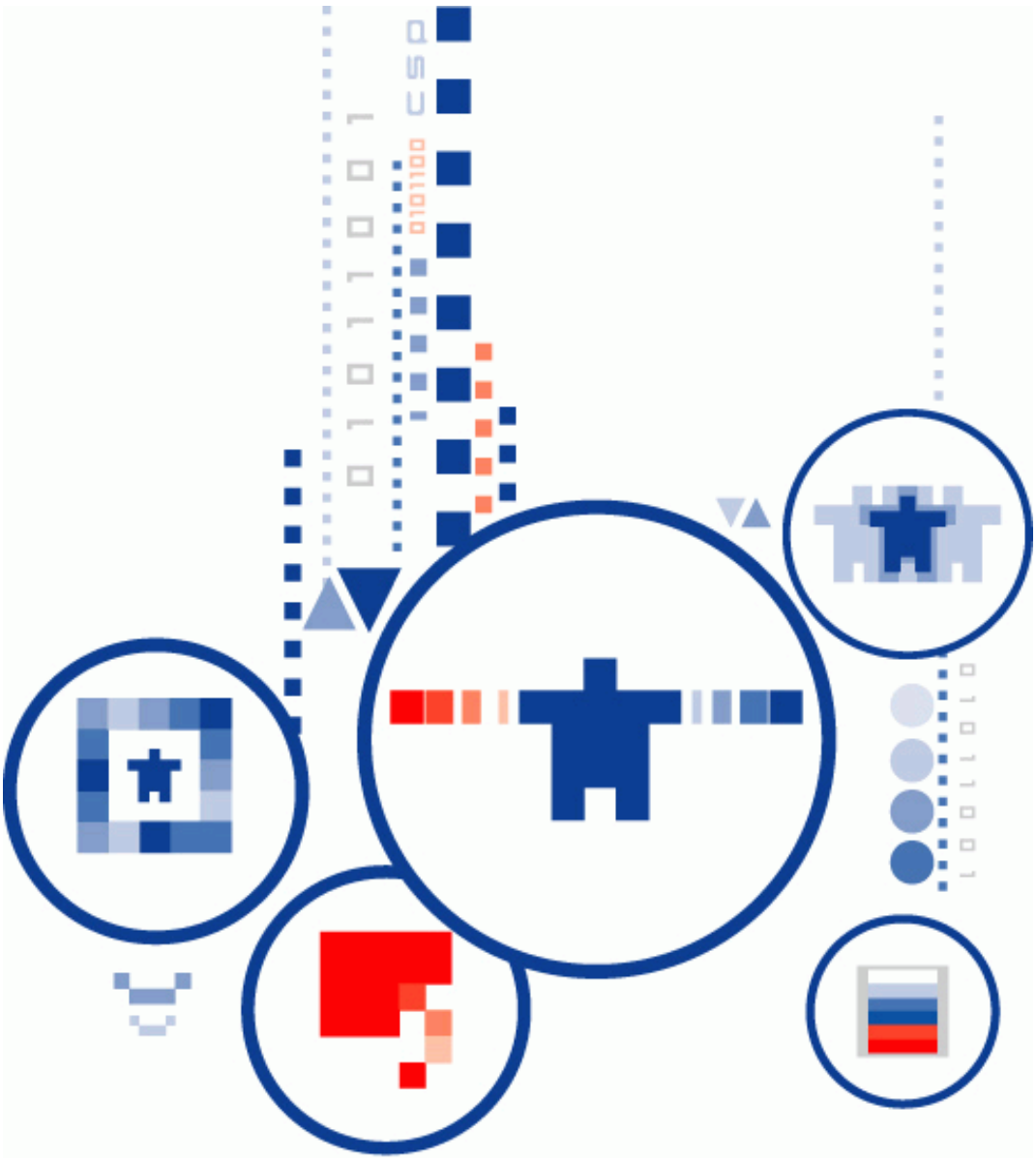


Программный комплекс

«КриптоПро Архив»

Руководство программиста

ЖТЯИ.00117-01 96 01

Данный документ представляет собой руководство для разработчика ПО, взаимодействующего с КриптоПро Архив (далее — Архив). Приведены примеры вызовов методов API Архива.

В компании КриптоПро развёрнут демонстрационный стенд Архива. Обратите внимание, доступ к демонстрационному стенду Архива осуществляется с помощью ключевой пары. Для получения доступа к демонстрационному стенду и онлайн-документации Swagger обратитесь в компанию КриптоПро по электронной почте info@cryptopro.ru.

На демонстрационном стенде расположено описание всех методов Архива: https://archive.cryptopro.ru:32764/swagger (ссылку открывать в Яндекс.Браузер или Chromium-Gost). Там же их можно опробовать. Адреса методов далее указаны относительно API демонстрационного стенда:

- client-api: https://archive.cryptopro.ru:32767 (предназначено для интеграции информационных систем),
- admin-api: https://archive.cryptopro.ru:32764 (предназначено только для информационных систем с правами администратора).

Некоторые функции доступны только начиная с определённых версий Архива. Для обозначения минимальной поддерживаемой версии используется специальный значок с указанием этой версии (например, v1.5.8+).

Оглавление

Оглавление 2
1 Внутреннее устройство 4
1.1 Контейнеры, хранилища и информационные системы 4
1.2 Состав контейнера электронного документа 5
1.3 Встроенные метаданные 5
1.4 Пользовательские метаданные 8
1.5 Возвращаемые значения 11
2 Создание контейнера 12

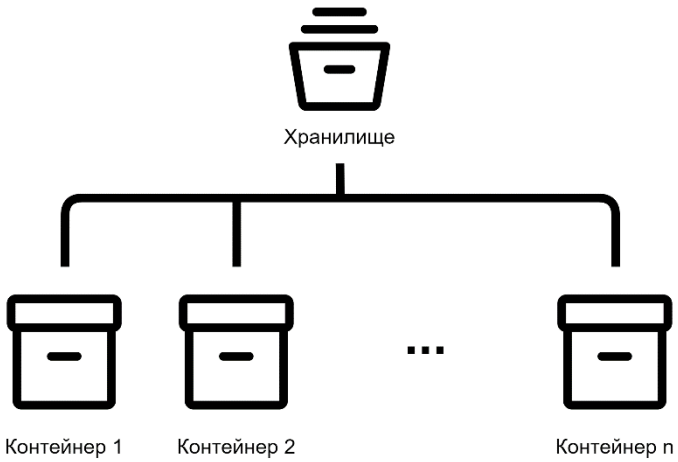
2.1 Общие доступные параметры 12
2.2 Примеры запросов cURL 14
3 Поиск контейнеров 15
3.1 Поиск по встроенным метаданным 15
3.2 Поиск по пользовательским метаданным 17
4 Получение подробной информации о контейнере 18
5 Получение статуса контейнера 19
5.1 Механизм уведомлений 20
5.2 Вызов базового метода API 21
5.3 Вызов расширенного метода API 22
6 Скачивание документов 22
6.1 Скачивание подписей в формате CAdES-A отдельно 22
6.2 Скачивание подписанного документа отдельно (требуется Архив УЦ) 23
6.3 Скачивание подписей и подписанного документа (требуется Архив УЦ) 24
7 Плагин подключения внешней СХД к подсистеме Архив УЦ 24
8 Плагин обработки подписи после усовершенствования 28

1 Внутреннее устройство

В данном разделе описано устройство Архива. Рассмотрены основные объекты, с которыми программисту необходимо работать при взаимодействии с Архивом.

1.1 Контейнеры, хранилища и информационные системы

Контейнер представляет собой совокупность подписи или нескольких подписей документа и набора метаданных, таких как, например, дата и время следующего усовершенствования этих подписей. Один контейнер может содержать в себе подписи только одного документа. Контейнер всегда содержит только усовершенствованные (последние актуальные) подписи документа: после очередного усовершенствования старые подписи уничтожаются. Контейнер не содержит в себе подписанный документ. Для хранения подписанных документов существует опциональная подсистема Архив УЦ. Если она активна, для сохранения документа достаточно передать соответствующий флаг (см. раздел 2 Создание контейнера). Контейнеры сохраняются в *хранилище*. Хранилищ может быть несколько.



Для работы с контейнерами и хранилищами информационной системе (разрабатываемой программе) необходимо иметь соответствующие права доступа: например, право поиска контейнеров, право создание контейнеров и т.д. в зависимости от требований, предъявляемых к информационной системе. Подробности о правах информационных систем можно найти в Руководстве администратора, глава 5 «Администрирование».

1.2 Состав контейнера электронного документа

Контейнер состоит из

- последних актуальных (усовершенствованных) подписей, которые автоматически усовершенствуются Архивом *,
- встроенных метаданных, которые всегда присутствуют у любого контейнера,
- произвольных пользовательских метаданных в формате словаря `Dictionary<string, string>`, которые администратор или информационная система может добавить для дальнейшего поиска по ним.

* в случае, если при усовершенствовании хотя бы одной подписи из контейнера произошла ошибка, Архив не будет совершать никаких автоматических действий по отношению к этому контейнеру, и его статус будет «Ошибка обработки подписи» или «Ошибка проверки подписи» в зависимости от типа произошедшей ошибки. Информационная система или администратор Архива могут отправить контейнер на повторную обработку, например, после устранения неполадки, которая привела к ошибке в процессе усовершенствования подписей в контейнере: например, восстановлен доступ к ранее недоступной службе TSP. Подробнее о том, как узнать статус контейнера, см. раздел 5 Получение статуса контейнера.

Контейнер **не** содержит подписанный документ. При необходимости хранения подписанных документов в связке с контейнером воспользуйтесь подсистемой Архив УЦ.

1.3 Встроенные метаданные

Ниже приведён список встроенных метаданных контейнера. Часть метаданных генерируется автоматически, часть задаётся пользователем. Названия метаданных ниже взяты из возвращаемого значения метода `GET /api/document/{id}`, возвращающего подробную информацию о контейнере.

Метаданные, актуальные для использования только для опциональной подсистемы Архив УЦ, отмечены значком Архив УЦ . Метаданные, которые устанавливаются или генерируются автоматически, отмечены значком AUTO .

Название	Описание
storage	Хранилище, в котором содержится контейнер. Задаётся при создании контейнера, в последствии **не** может быть изменено. Пример значения поля: `{"id": 1, "name": "Бухгалтерия"}`.
sourceSystem	AUTO Информационная система, к которой привязан контейнер. В значении поля устанавливается информационная система, создавшая контейнер. Пример: `{"id": 1, "name": "Главный бухгалтер"}`.
useArchiveCa	Архив УЦ Флаг, указывающий, сохранён ли подписанный документ в опциональной подсистеме Архив УЦ. **ОБРАТИТЕ ВНИМАНИЕ**: Этот флаг **устарел** и не должен использоваться в новых приложениях. Вместо него определяйте статус подписанного документа с помощью параметра `signedDocumentStatus`.
signedDocumentStatus	v1.5.5+ Архив УЦ AUTO Статус подписанного документа: • `0` — подписанный документ не отслеживается подсистемой Архив УЦ, • `1` — подписанный документ доступен для скачивания, • `2` — подписанный документ отслеживается подсистемой Архив УЦ, но не доступен для скачивания. Актуальный список возможных значений параметра для данной версии Архива можно получить, вызвав метод `GET /api/dictionary/signeddocuments`.
documentType	Архив УЦ AUTO MIME-тип подписанного документа. Заполняется автоматически на основе полученного подписанного документа. С случае, если определить тип невозможно, поле устанавливается в значение `null` — в этом случае считается, что MIME-тип подписанного документа — `application/octet-stream`.
saveType	Тип хранения подписей: • `1` — временное хранение. В этом случае срок хранения обязательно должен быть задан и определяется параметром `archiveUntilDate`, • `2` — постоянное хранение. В этом случае значение поля `archiveUntilDate` не играет роли.

	Актуальный список возможных значений параметра для данной версии Архива можно получить, вызвав метод `GET /api/dictionary/savetypes`.
structureName	Имя структурного подразделения, создавшего контейнер. Пример: `"Отдел 1"`.
modifiedDate	AUTO Дата последнего изменения контейнера. Пример: `"2023-05-30T20:00:35.17556+03:00"`.
createDate	AUTO Дата создания контейнера. Пример: `"2023-05-30T20:00:35.17556+03:00"`.
signatureUpdateDate	AUTO Дата истечения срока действия последнего архивного штампа времени. Присутствует только у контейнеров, которые уже были усовершенствованы хотя бы один раз. Пример: `"2023-05-30T20:00:35.17556+03:00"`.
archiveUntilDate	Дата окончания архивного хранения контейнера. Пример: `"2023-05-30T20:00:35.17556+03:00"`.
id	AUTO Уникальный идентификатор контейнера. Генерируется при создании контейнера и возвращается в теле ответа. Пример: `"bc32d0e9-d085-4236-9e46-ba88d0490a41"`.
name	Имя контейнера. Не обязано быть уникальным. Пример: `"Мой контейнер"`.
status	AUTO Статус контейнера: • 2 — ошибка обработки подписи. В этом статусе находятся контейнеры, усовершенствование подписей которых завершилось ошибкой в связи с тем, что внешние службы, необходимые для усовершенствования, не были доступны, • 3 — удалён, • 4 — архивное хранение, • 5 — ошибка проверки подписи. В этом статусе находятся контейнеры, усовершенствование подписей которых завершилось с ошибкой в связи с тем, что хотя бы одна подпись оказалась недействительной, • 6 — подготовка к хранению. В этом статусе находятся контейнеры, которые ещё не были обработаны, • 7 — в процессе обработки, • 8 — требуется обновление подписи. Этот статус является сигналом Архиву, что подписи в данном

	контейнере необходимо усовершенствовать. Переход в этот статус может произойти как автоматически, когда пришло время повторного усовершенствования подписей, так и вручную вызовом метода `POST /api/document/{id}/retry`. Актуальный список возможных значений параметра для данной версии Архива можно получить, вызвав метод `GET /api/dictionary/statuses`.
`isArchived`	AUTO Флаг, указывающий, находится ли контейнер на постоянном хранении. В случае, если значение флага false, Архив не отслеживает данный контейнер.

Если какое-то поле не указано в перечне выше, оно оставлено для обратной совместимости. Среди таких полей:

- `hashes`,
- `signatures`,
- `isCreateDate`,
- `isAvailable`.

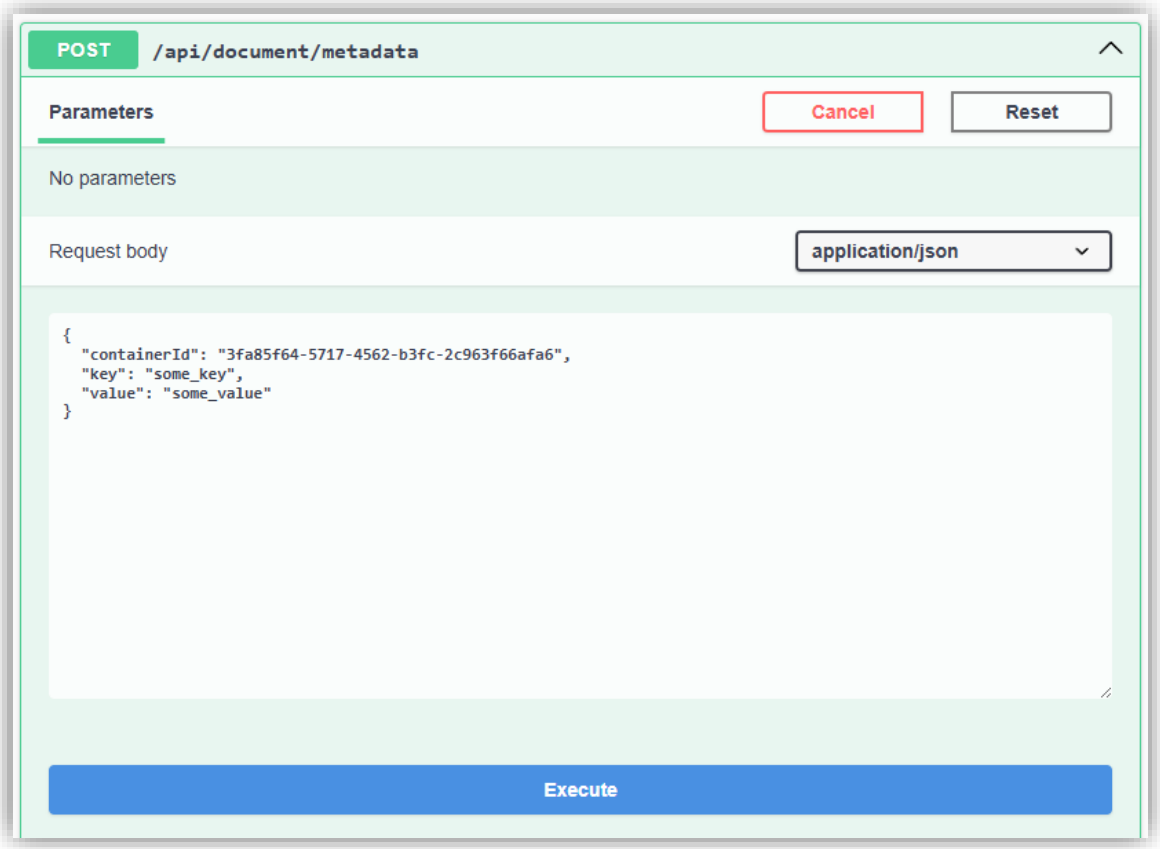
1.4 Пользовательские метаданные v1.5.8+

Пользователь может добавить, удалить или изменить произвольные метаданные в формате ключ/значение (`Dictionary<string, string>`). Добавить пользовательские метаданные можно при создании контейнера или к уже созданному контейнеру. Доступен поиск контейнеров по пользовательским метаданным (см. раздел 3.2 Поиск по пользовательским метаданным).

Добавить пользовательские метаданные к созданному контейнеру можно методом `POST /api/document/metadata`. Параметры запроса:

Параметр	Описание
`containerId`	Идентификатор контейнера, к которому необходимо добавить запись метаданных.
`key`	Ключ добавляемого поля метаданных. Пример: `"some_key"`.
`value`	Значение добавляемого поля метаданных. Пример: `"some_value"`.

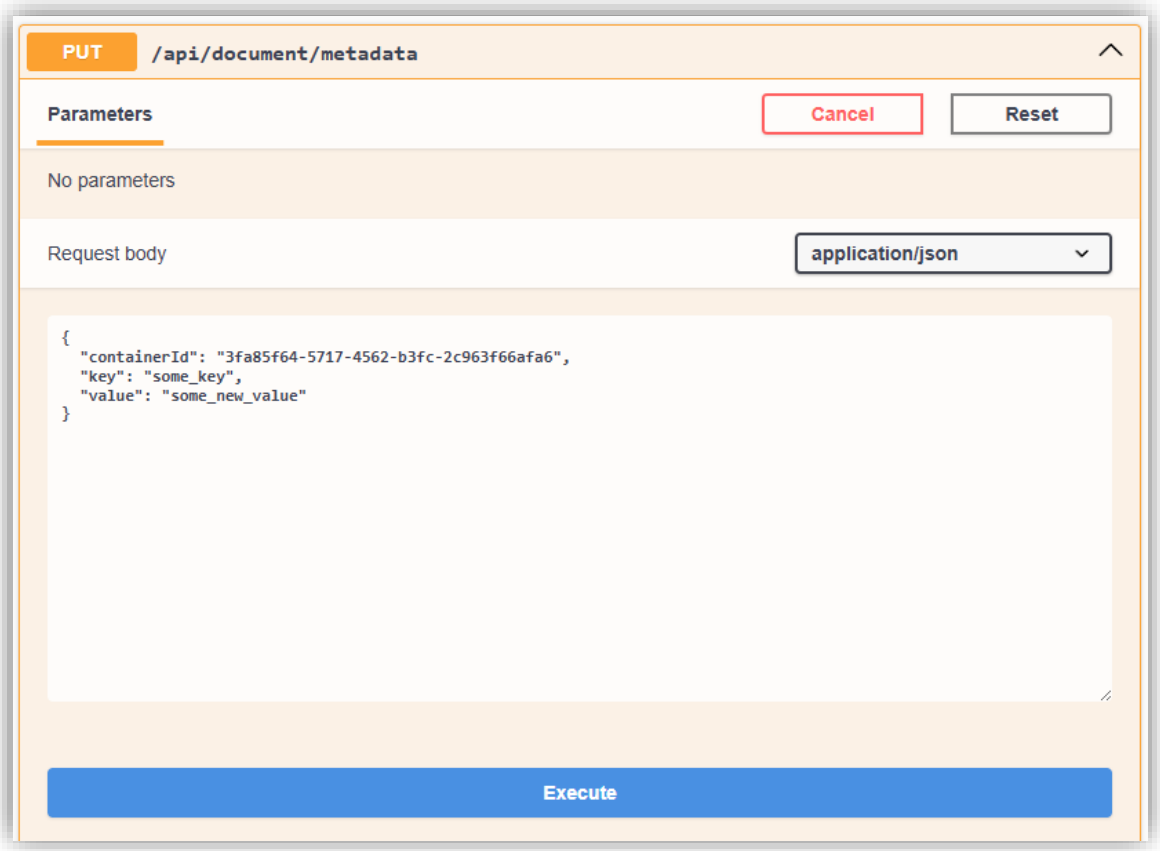
Пример заполнения параметров запроса в Swagger:



Изменить пользовательские метаданные к созданному контейнеру можно методом `PUT /api/document/metadata`. Параметры запроса:

Параметр	Описание
`containerId`	Идентификатор контейнера, у которого необходимо изменить запись метаданных.
`key`	Ключ изменяемого поля метаданных. Пример: `"some_key"`.
`value`	Новое значение поля метаданных. Пример: `"some_new_value"`.

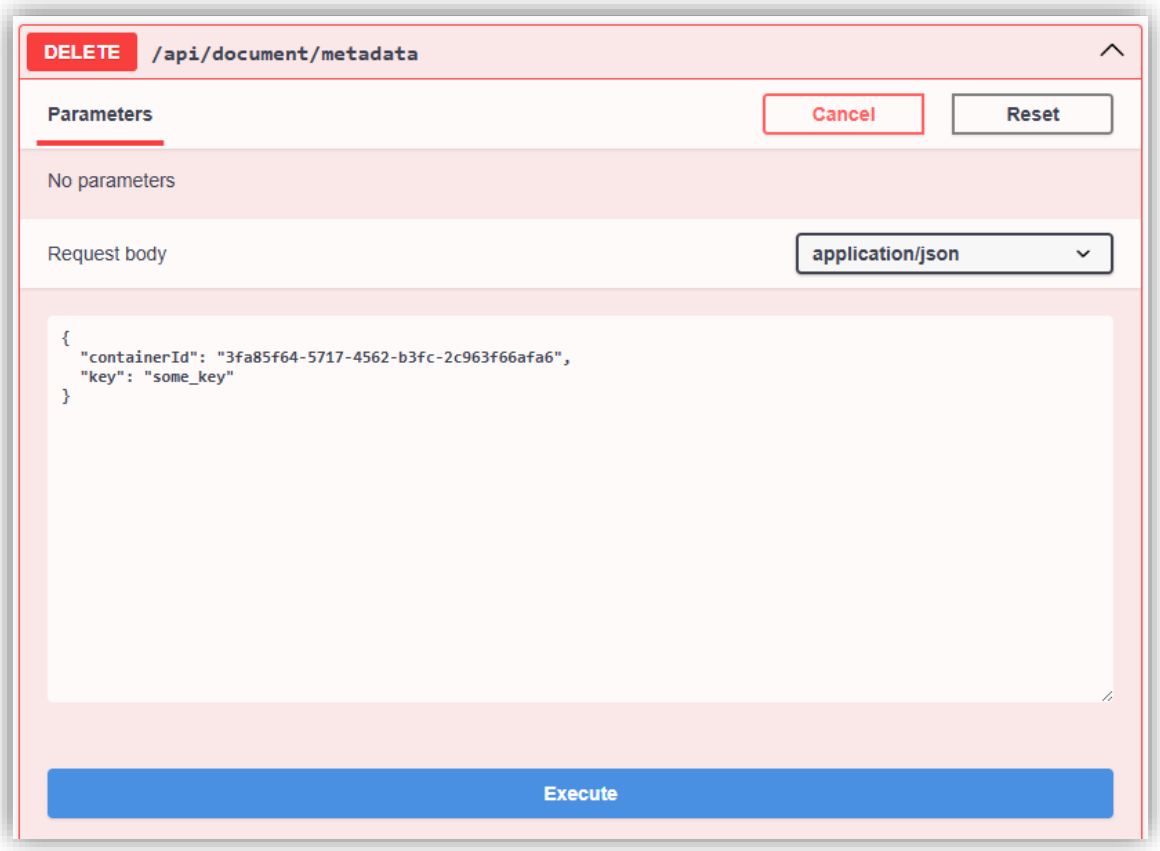
Пример заполнения параметров запроса в Swagger:



Удалить пользовательские метаданные к созданному контейнеру можно методом `DELETE /api/document/metadata`. Параметры запроса:

Параметр	Описание
`containerId`	Идентификатор контейнера, у которого необходимо удалить запись метаданных.
`key`	Ключ поля метаданных, которое нужно удалить. Пример: `"some_key"`.

Пример заполнения параметров запроса в Swagger:



Для получения справки о добавлении метаданных при создании контейнера обратитесь к разделу о создании контейнера.

1.5 Возвращаемые значения

В Архиве используется единый шаблон ответа, имеющий следующий вид:

```
{
    "data": <возвращаемый объект или null в случае ошибки>,
    "errorCode": <код ошибки Архива или null в случае успеха>,
    "errorDescription": <текст ошибки Архива или null в случае успеха>
}
```

Пример ответа с ошибкой:

```
{
    "data": null,
    "errorCode": "DocumentNotAvailable",
    "errorDescription": "Документ не существует или недоступен."
}
```

Пример успешного ответа:

```
{
    "data": 4,
    "errorCode": null,
    "errorDescription": null
}
```

В зависимости от результата выполнения запроса Архив устанавливает в ответе соответствующий код состояния HTTP. Возможные коды:

- в случае успеха возвращаемый код состояния HTTP будет иметь значение из диапазона 2XX,
- в случае ошибки, связанной с неправильными параметрами запроса, возвращаемый код будет иметь значение из диапазона 4XX. Примеры таких ошибок:
  - недостаточно прав для выполнения операции (403),
  - контейнер не найден в системе (404),
  - параметр задан неверно (400);
- в случае ошибки на стороне сервера возвращаемый код будет иметь значение из диапазона 5XX. При получении такой ошибки необходимо смотреть журнал работы Архива. Например, такая ошибка может возникнуть в случае отсутствия подключения к базе данных.

2 Создание контейнера

Для создания контейнера существуют методы:

- `POST /api/document/content` — по исходному документу,
- `POST /api/document/hash` — по хэш-сумме исходного документа.

2.1 Общие доступные параметры

Тело запроса имеет тип `multipart/form-data` и состоит из следующих полей, приведённых в таблице ниже. Метаданные, актуальные только для опциональной подсистемы Архив УЦ, отмечены значком Архив УЦ .

Имя поля	Описание
Content (обязательно) (только при создании по исходному документу)	Бинарное содержимое подписанного файла. В случае присоединённой подписи — сама подпись.
DocumentHash (обязательно) (только при создании по хэш-сумме)	Хэш-сумма подписанного документа.
Signs	Список бинарных содержимых подписей файла. В случае присоединённой подписи поле не передавать. Подписи в кодировке `base64` не принимаются.
StorageId (обязательно)	Идентификатор хранилища, в котором необходимо создать контейнер. Список доступных хранилищ с идентификаторами можно получить методом `GET /api/user/storages`.
SaveType (обязательно)	Тип хранения контейнера. Может быть временным (Temporary) или постоянным (Permanent). В случае временного типа хранения необходимо также передать дату окончания хранения в параметре **ArchiveDate**.
ContainerName (обязательно)	Имя контейнера.
StructureName	Имя структурного подразделения.
ArchiveDate	Дата окончания хранения. Обязательна в случае, если в параметре **SaveType** выбран тип хранения временный (Temporary).
IsNeedArchivatorSign	Не используемый флаг, оставленный для обратной совместимости. Не передавать.
UseArchiveCa (только при создании по исходному документу)	Архив УЦ Флаг требования использования модуля Архив УЦ для сохранения в нём подписанного документа.
ArchiveCaPluginName (только при создании по исходному документу)	Архив УЦ Уникальное имя плагина подсистемы Архив УЦ. Если не указан, документ будет сохранён в базе данных подсистемы Архив УЦ.

Metadata	v1.5.8+ Словарь с метаданными. Данное поле можно использовать для хранения произвольных метаданных в формате ключ/значение. Пример: {"some_key": "some_value", "another_key": "another_value"}.

ПРИМЕЧАНИЕ

Для новых программ рекомендуется использование именно механизма пользовательских метаданных, так как он оптимизирован для быстрого поиска.

2.2 Примеры запросов cURL

Ниже приведён пример `curl`-запроса для создания контейнера. Обратите внимание, что используется `curl` из пакета CryptoPro CSP, который поддерживает ГОСТ TLS — обычный `curl` не подойдёт. Выделенные поля обязательно необходимо заменить на свои значения.

```
/opt/cprocsp/bin/amd64/curl \
    # Отпечаток сертификата пользователя.
    -E '7db3643bf1f651831a65a503409ec59e79a7bb9f' \
    -X 'POST' \
    'https://archive.cryptopro.ru:32764/api/document/content' \
    -H 'accept: text/plain' \
    -H 'Content-Type: multipart/form-data' \
    -F 'ContainerName=Test' \
    # Идентификатор хранилища, в котором создаётся контейнер.
    -F 'StorageId=1' \
    -F 'SaveType=Permanent' \
    # Путь к подписанному документу или присоединённой подписи.
    -F 'Content=@./document.txt;type=text/plain' \
    -F 'UseArchiveCa=false' \
    -F 'Metadata={"some_field": "some_value"}' \
    # Пути к присоединённым подписям. Могут отсутствовать, если подпись присоединённая.
    -F 'Signs=@./document.txt.1.sig;type=application/sig' \
    -F 'Signs=@./document.txt.2.sig;type=application/sig'
```

Пример ответа с комментариями:

```
{
    "data": "bc32d0e9-d085-4236-9e46-ba88d0490a41", // идентификатор созданного контейнера
    "errorCode": null, // код ошибки
    "errorDescription": null // текст ошибки
}
```

В поле `data` находится идентификатор созданного контейнера. По нему можно получать доступ к информации о контейнере и к подписям контейнера.

3 Поиск контейнеров

В Архиве есть несколько способов искать контейнеры:

- поиск по встроенным метаданным контейнера (хранилище, статус и т.д., см. раздел 1.3 Встроенные метаданные),
- поиск по произвольным пользовательским метаданным контейнера (подробности см. в разделе 1.4 Пользовательские метаданные).

ПРИМЕЧАНИЕ

Для новых программ рекомендуется использование именно механизма пользовательских метаданных, так как он оптимизирован для быстрого поиска.

3.1 Поиск по встроенным метаданным

Получить идентификаторы уже существующих контейнеров можно методом `POST /api/document/search`. Тело запроса состоит из полей **`filters`**, **`pageSize`** и **`page`**.

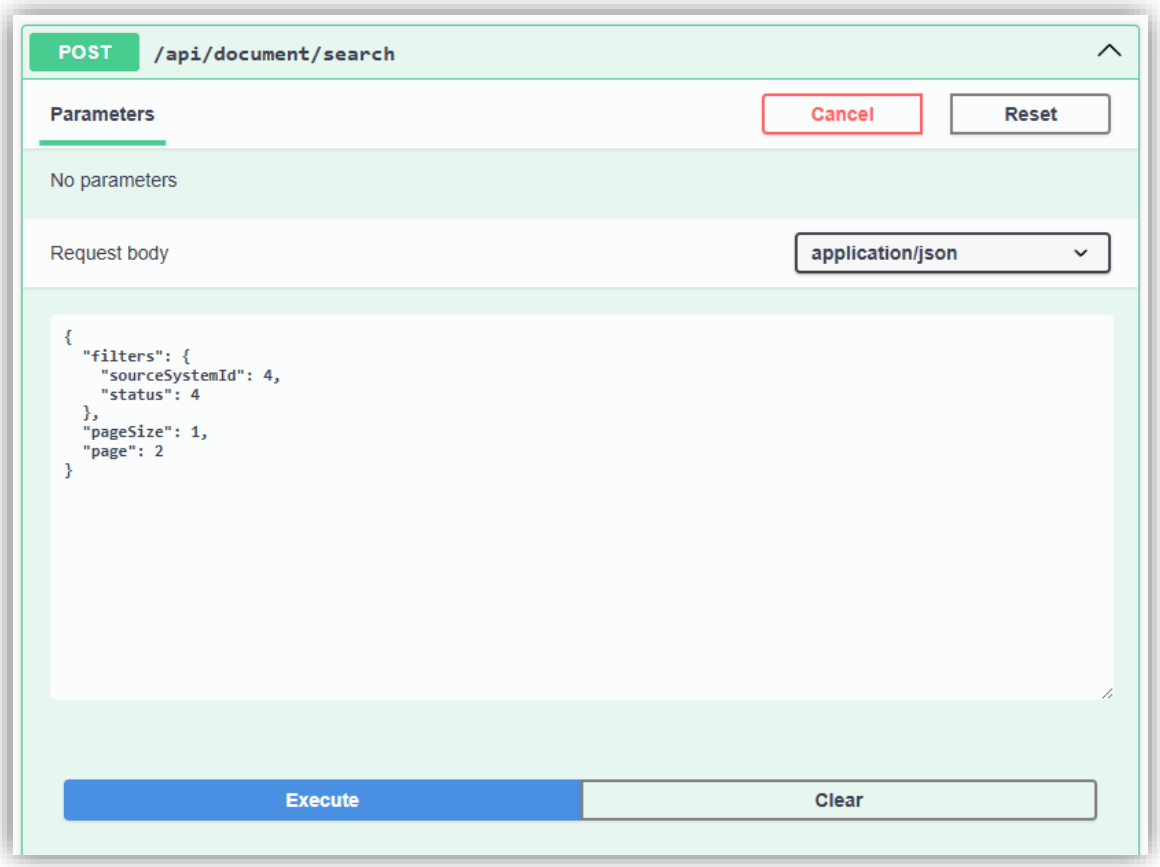
В **`filters`** указываются фильтры поиска. Если какой-то из фильтров не требуется применять, его не следует указывать в запросе. Если никакие фильтры не требуется применять, поле **`filters`** можно опустить целиком. Доступные фильтры:

Имя фильтра	Пример	Описание
`storageIds`	`[1, 2, 3]`	Список идентификаторов хранилищ, в которых проводить поиск.
`name`	`"MyContainer"`	Подстрока имени контейнера.
`sourceSystemId`	`7`	Идентификатор пользователя, загрузившего контейнер.
`status`	`4`	Статус контейнера.
`isArchived`	`true`	Флаг, указывающий на то, что контейнер находится на архивном хранении. Если передать значение **`false`**, будут показаны

только контейнеры с типом хранения **Временный**, хранение которых завершено.

pageSize (по умолчанию 10) — количество контейнеров, которые нужно вернуть на одной странице. **page** (по умолчанию 1) — номер страницы. Нумерация начинается с единицы. Соответственно, при отправке запроса с пустым телом будут возвращены последние 10 добавленных контейнеров.

Пример заполнения формы в Swagger для поиска контейнеров в статусе **Архивное хранение**, которые создал пользователь с идентификатором **4**. Запрашивается вторая страница с одним контейнером на странице:



Пример ответа с комментариями:

```
{
    "data": {
        "documents": [ // список контейнеров на запрашиваемой странице
            {
                "id": "bc32d0e9-d085-4236-9e46-ba88d0490a41", // идентификатор контейнера
                "name": "MyContainer", // имя контейнера
                "status": 4, // статус контейнера
                "isArchived": true // флаг, указывающий, что контейнер находится на архивном
                                   хранении
            }
        ],
        "totalRecords": 11 // общее количество найденных контейнеров
    },
    "errorCode": null,
    "errorDescription": null
}
```

3.2 Поиск по пользовательским метаданным v1.5.8+

Данный вид поиска позволяет искать контейнеры по произвольным пользовательским метаданным. Для получения списка контейнеров, содержащих данное значение пользовательских метаданных, предназначен метод `POST /api/document/search/metadata`. Пример вызова метода приведён ниже. Выделенные значения параметров в примере необходимо обязательно заменить на свои значения.

```
/opt/cprocsp/bin/amd64/curl \
    -E '7db3643bf1f651831a65a503409ec59e79a7bb9f' \
    -X POST \
    'https://pki-cluster-core/api/document/search/metadata' \
    -H 'accept: text/plain' \
    -H 'Content-Type: application/json' \
    -d '{
        "key": "some_field",
        "value": "some_value"
    }'
```

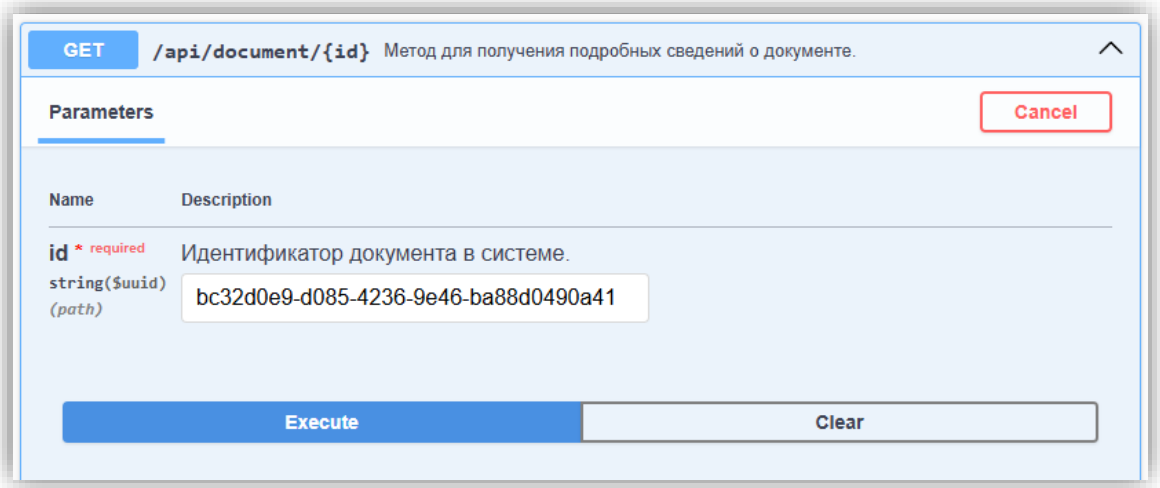
Пример ответа:

```
{
    "data": {
        "containers": [
            {
                "storageId": 1,
                "metadata": {
                    "some_field": "some_value",
                    "ExternalId": "0299c8e2-539b-49bb-ba03-3479f7b8085a",
                    "RelatedFile": "/var/lib/program/files/somefile.pdf"
                },
                "id": "8731e50f-c444-4c30-9ebb-4a45d7d3ce1c",
                "name": "metadata_test",
                "status": 6,
                "isArchived": true
            }
        ]
    },
    "errorCode": null,
    "errorDescription": null
}
```


Данный способ поиска контейнеров имеет ограничение: возможно искать только по одному значению: нельзя указать несколько разных значений пользовательских метаданных контейнера.

4 Получение подробной информации о контейнере

Для получения информации о контейнере предназначен метод `GET /api/document/{id}`. Метод возвращает подробную информацию о контейнере. В случае, если контейнер отсутствует в системе, метод вернёт ошибку **404** (**Not found**). Пример заполнения параметров запроса в Swagger:



Пример ответа с комментариями ниже. Для более подробной справки по каждому полю обратитесь к разделу о встроенных метаданных:

```
{
    "data": {
        "storage": { // хранилище, в котором находится контейнер
            "id": 4,
            "name": "Техническое сопровождение"
        },
        "sourceSystem": { // пользователь, создавший контейнер
            "id": 4,
            "name": "Служба технического сопровождения"
        },
        "hashes": null, // поле оставлено для обратной совместимости
        "signatures": null, // поле оставлено для обратной совместимости
        "isToBeSigned": false, // поле оставлено для обратной совместимости
        "useArchiveCa": false, // доступен ли подписанный документ в подсистеме Архив УЦ
        "signedDocumentStatus": 0, // статус подписанного документа в подсистеме Архив УЦ
        "documentType": "text/plain", // MIME-тип подписанного документа
        "saveType": 1, // тип хранения (в данном случае Временный)
        "structureName": null, // имя структурного подразделения
        "modifiedDate": "2023-05-30T20:00:35.17556+03:00", // дата последнего изменения
        "isCreateDate": "2023-05-30T20:00:32.183581+03:00", // поле оставлено для обратной
                                                               совместимости
        "createDate": "2023-05-30T20:00:32.18363+03:00", // дата создания контейнера
        "signatureUpdateDate": "2024-01-05T15:50:05+03:00", // дата добавления следующего
                                                               архивного штампа к сохранённым подписям
        "archiveUntilDate": "2023-07-21T00:00:00+03:00", // дата окончания хранения
        "isAvailable": true,
        "metadata": { // пользовательские произвольные метаданные
            "some_key": "some_value"
        },
        "id": "bc32d0e9-d085-4236-9e46-ba88d0490a41", // идентификатор контейнера
        "name": "MyContainer", // имя контейнера
        "status": 4, // статус контейнера
        "isArchived": true
    },
    "errorCode": null,
    "errorDescription": null
}
```


Подробности о составе контейнера, см. в разделе 1.2 Состав контейнера электронного документа.

5 Получение статуса контейнера

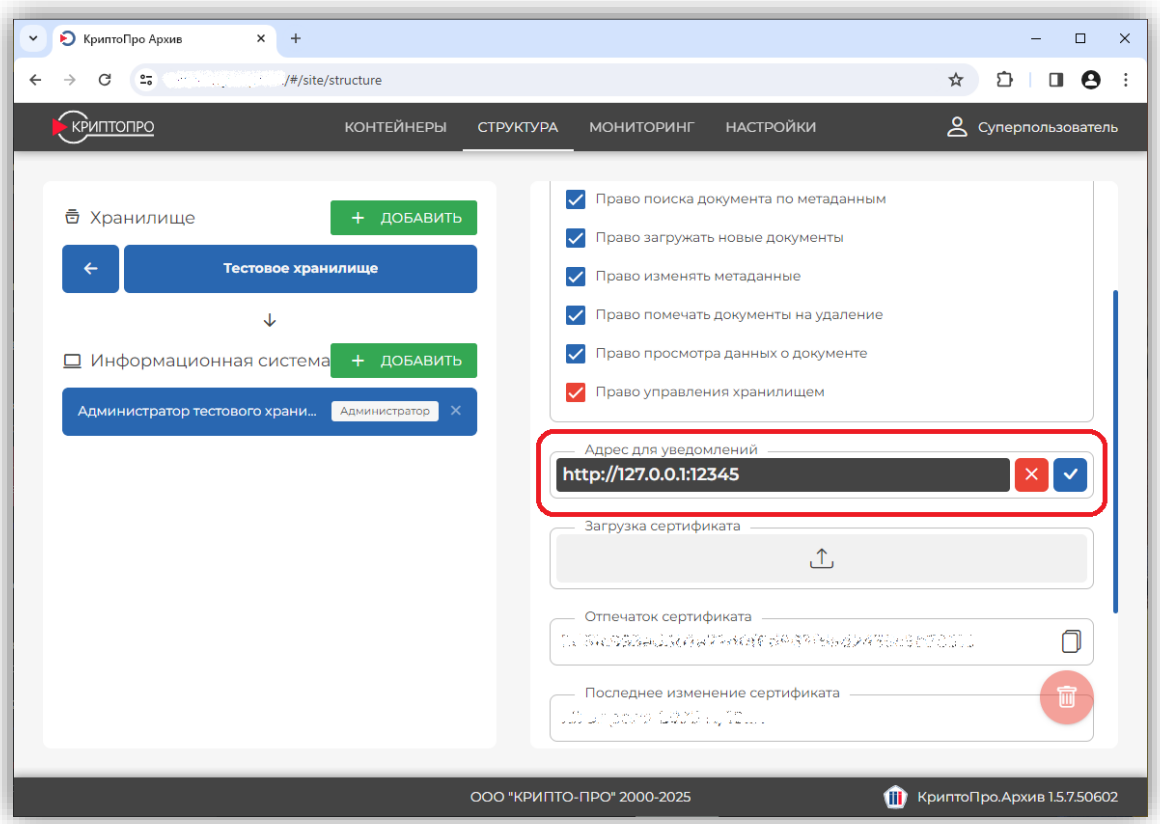
Существует три способа получения статуса контейнера:

- с помощью механизма уведомлений (рекомендуемый для большинства сценариев),
- с помощью вызова метода `GET /api/document/{id}/status`,
- v1.5.13+ расширенная информация с помощью вызова метода `GET /api/document/{id}/status_ext`.

Ниже рассмотрены все способы.

5.1 Механизм уведомлений

В Архиве есть возможность получить уведомление об изменении статуса контейнера после его усовершенствования с гарантированной доставкой. Для этого у информационной системы, от лица которой планируется создавать контейнеры, необходимо заполнить поле «Адрес для уведомлений», которое выделено на скриншоте ниже.



Уведомления об изменении статусов контейнеров, созданных данной информационной системой, будут отправляться на указанный адрес.

Уведомление представляет собой POST-запрос со следующим телом:

```
{
  "containerId": "<containerId>"
  "newStatus": <newStatus>
}
```

Описание полей:

Поле	Тип	Описание
containerId	Строка	Уникальный идентификатор контейнера
newStatus	Число	Число, соответствующее новому статусу контейнера. Для получения соответствия цифры статуса его имени используйте метод GET /api/dictionatry/statuses

Пример тела уведомления:

```
{
  "containerId": "01e2f572-fa2c-44c3-be78-f7e32893cc83"
  "newStatus": 4
}
```

На этот запрос необходимо ответить успешным HTTP-кодом из диапазона 2XX. В случае, если код ответа будет не успешен или уведомление не удастся отправить, уведомление не будет считаться отправленным, и оно будет отправлено повторно позже. Количество времени, через которое уведомление будет отправлено повторно, можно указать в настройках программы consumer (параметр DelayMs, см. раздел 3.8 Руководства администратора). В случае успешной отправки уведомления в истории контейнера будет оставлена соответствующая запись.

5.2 Вызов базового метода API

ВАЖНО: данный способ крайне не рекомендуется использовать в случае, когда необходимо отслеживать статус контейнера и реагировать на его изменения. В таком сценарии используйте механизм уведомлений.

Если необходимо запросить только статус контейнера, используйте метод GET /api/document/{id}/status. Вызов этого метода аналогичен вызову метода GET /api/document/{id}. В поле data ответа будет возвращено число, соответствующее статусу контейнера. Для получения соответствия цифры статуса его имени используйте метод GET /api/dictionatry/statuses.

В случае, если контейнер отсутствует в системе, метод вернёт ошибку **404** (**Not found**).

5.3 Вызов расширенного метода API v1.5.13+

ВАЖНО: данный способ крайне не рекомендуется использовать в случае, когда необходимо отслеживать статус контейнера и реагировать на его изменения. В таком сценарии используйте механизм уведомлений.

Если необходимо запросить статус контейнера и каждой подписи в нём, используйте метод `GET /api/document/{id}/status_ext`. Вызов этого метода аналогичен вызову метода `GET /api/document/{id}`. В поле `data` ответа будет возвращён объект, описывающий статус контейнера и его подписей:

```
{
  "data": {
    "containerStatus": 4,
    "signatureStatuses": [
      {
        "id": 123,   // уникальный идентификатор подписи
        "status": 4  // статус подписи (возможные значения те же, что у статуса контейнера)
      },
      ...
    ]
  },
  ...
}
```

Для получения соответствия цифры статуса его имени используйте метод `GET /api/dictionatry/statuses`.

В случае, если контейнер отсутствует в системе, метод вернёт ошибку **404** (**Not found**).

6 Скачивание документов

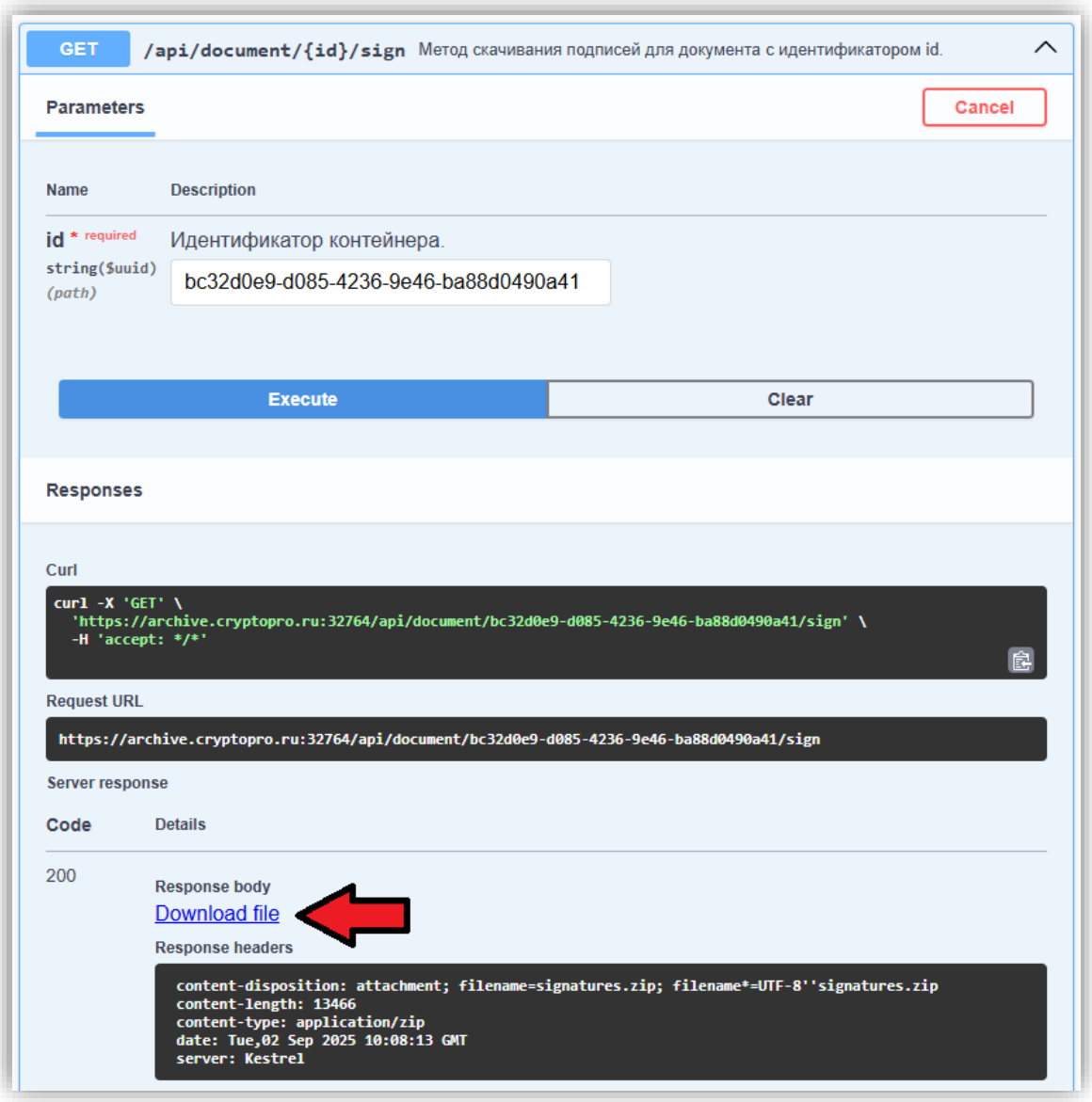
В Архиве предусмотрена возможность скачать как подписи отдельно, так и подписи вместе с подписанным документом в случае использования Архива УЦ для хранения подписанного документа данного контейнера.

6.1 Скачивание подписей в формате CAdES-A отдельно

Для скачивания подписей, сохранённых в контейнере, используйте метод `GET /api/document/{id}/sign`. Если вы загрузили подписи, но они не успели усовершенствоваться, метод вернёт те неусовершенствованные подписи, которые вы загрузили. Индикатором того, что подписи были усовершенствованы,

служит статус контейнера **Архивное хранение**. Метод возвращает zip-архив с подписями. Тип содержимого ответа — application/zip.

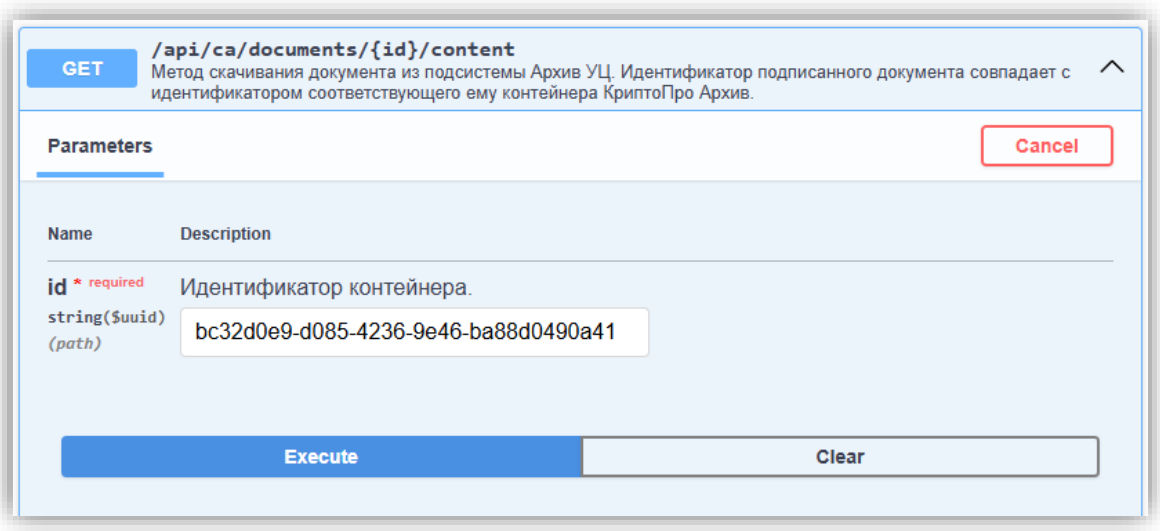
После выполнения в Swagger появится ссылка на скачивание (**Download file**):



Для скачивания нажмите на неё.

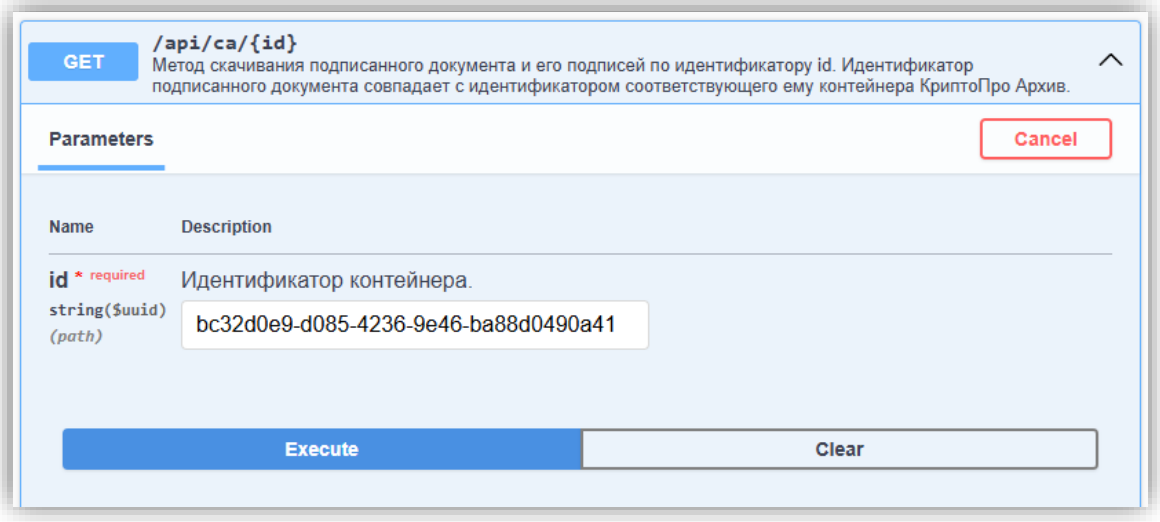
6.2 Скачивание подписанного документа отдельно (требуется Архив УЦ)

Для скачивания только подписанного документа используется метод GET /api/ca/documents/{id}/content, где {id} — идентификатор контейнера в Архиве. Ниже приведён пример заполнения параметров запроса в Swagger.



6.3 Скачивание подписей и подписанного документа (требуется Архив УЦ)

Для скачивания подписанного документа вместе с подписями используется метод `GET /api/ca/{id}`, где `{id}` — идентификатор контейнера в Архиве. Метод возвращает `zip`-архив с подписями и подписанным документом. Тип содержимого ответа — `application/zip`. Ниже приведён пример заполнения запроса в Swagger.



7 Плагин подключения внешней СХД к подсистеме Архив УЦ

В данном разделе описан механизм создания плагина подключения внешней СХД к подсистеме Архив УЦ. В качестве примера рассмотрим процесс написания плагина, сохраняющего подписанные документы на жёсткий диск.

Плагин представляет собой динамическую библиотеку, написанную на языке C#, в которой содержится реализация интерфейса `IDocumentStoragePlugin` из библиотеки `CryptoPro.Archive.Plugins.Design`.

Плагин необходимо расположить в папке `cp-archive/plugins` (она расположена рядом с папками `admin-api`, `client-api` и так далее). По умолчанию это `/opt/cp-archive/plugins` на UNIX и `C:\inetpub\cp-archive\plugins` на Windows. Плагин используется программой `document-consumer` — убедитесь, что она установлена.

Перейдём к примеру написания собственного плагина. Для начала создайте C#-проект и в конфигурационном файле `.csproj` укажите следующие настройки:

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <TargetFramework>net6.0</TargetFramework>
        <ImplicitUsings>enable</ImplicitUsings>
        <Nullable>enable</Nullable>
        <EnableDynamicLoading>true</EnableDynamicLoading>
    </PropertyGroup>
    <ItemGroup>
        <Reference Include="CryptoPro.Archive.Plugins.Design">
            <HintPath>path/to/CryptoPro.Archive.Plugins.Design.dll</HintPath>
            <Private>false</Private>
            <ExcludeAssets>runtime</ExcludeAssets>
        </Reference>
    </ItemGroup>
</Project>
```

Выделены важные строки. Замените значение `HintPath` из секции `ItemGroup` на путь к скачанной библиотеке `CryptoPro.Archive.Plugins.Design.dll`.

Далее реализуйте интерфейс `IDocumentStoragePlugin`. Ниже приведён пример плагина, сохраняющего документы на жёсткий диск.

```
using CryptoPro.Archive.Plugins.Design.Interfaces;
using CryptoPro.Archive.Plugins.Design.Models;

namespace FilesystemStoragePlugin;

/// <summary>
/// Пример плагина, позволяющего хранить документы подсистемы Архив УЦ в файловой системе.
/// </summary>
public class FilesystemStoragePlugin : IDocumentStoragePlugin
{
    private string _documentsFolderPath;

    /// <inheritdoc/>
    public void Initialize(Dictionary<string, string> parameters)
    {
        var parametersIgnoreCase = new Dictionary<string, string>(
            parameters, StringComparer.InvariantCultureIgnoreCase);
        _documentsFolderPath = parametersIgnoreCase["DocumentsFolderPath"];
    }

    /// <inheritdoc/>
    public Result<string> Save(Container container)
    {
        var filepath = GenerateFilepath(container.Id);

        try
        {
            File.WriteAllBytes(filepath, container.Document.Raw);
            return filepath;
        }
        catch (Exception exception)
        {
            return Result.Failure<string>(
                $"Unable to save document. Error message: \"{exception.Message}\".");
        }
    }

    /// <inheritdoc/>
    public Result<Document> Get(string path)
    {
        if (!File.Exists(path))
        {
            return Result.Failure<Document>($"File not found. Path: \"{path}\".");
        }

        try
        {
            var raw = File.ReadAllBytes(path);
            return new Document(raw);
        }
        catch (Exception exception)
        {
            return Result.Failure<Document>(
                $"Unable to get document. Error message: \"{exception.Message}\".");
        }
    }

    /// <inheritdoc/>
```

```
    public Result Delete(string path)
    {
        if (!File.Exists(path))
        {
            return Result.Failure($"Unable to delete file \"{path}\". File not found.");
        }

        try
        {
            File.Delete(path);
            return Result.Success();
        }
        catch (Exception exception)
        {
            return Result.Failure(
                $"Unable to delete file \"{path}\". Error: \"{exception.Message}\".");
        }
    }

    /// <summary>Сгенерировать путь к файлу на основе идентификатора контейнера.</summary>
    /// <param name="containerId">Идентификатор контейнера.</param>
    /// <returns>Путь к файлу.</returns>
    private string GenerateFilepath(Guid containerId) =>
        Path.Combine(_documentsFolderPath, containerId.ToString());
}
```

Плагин не должен генерировать исключения. Если во время выполнения какой-либо операции произошла ошибка, используйте класс `Result` для передачи информации об этой ошибке в вызывающую функцию.

Также обратите внимание, что при создании плагина будет вызван конструктор без параметров. Для инициализации плагина используйте функцию `void Initialize(Dictionary<string, string> parameters)`. Параметр `parameters` будет прочитан из секции `Plugins` конфигурационного файла `plugin-config.json` из папки с плагинами `cp-archive/plugins`. Если файл отсутствует, создайте его. Пример содержимого конфигурационного файла для плагина выше:

```
{
  "Plugins": {
    "DocumentStorage": [
      {
        "IsEnabled": true,
        "ReadOnly": false,
        "PluginName": "filesystem_01",
        "AssemblyName": "FilesystemStoragePlugin.dll",
        "Parameters": {
          "DocumentsFolderPath": "/home/username/documents"
        }
      }
    ]
  }
}
```

Секция `Plugins.DocumentStorage` содержит список, что позволяет подключить несколько плагинов одновременно. Параметр `PluginName` содержит имя плагина, которое используется для его идентификации в дальнейшем. Имя должно быть уникальным и непустым.

Выбор плагина при создании контейнера осуществляется при вызове `POST /api/document/content` с помощью указания имени плагина в необязательном параметре `ArchiveCaPluginName` метода.

8 Плагин обработки подписи после усовершенствования

В данном разделе описан механизм создания плагина обработки подписей после усовершенствования. В качестве примера рассмотрим процесс написания плагина, сохраняющего подписи на жёсткий диск и создающего небольшой отчёт об усовершенствовании для каждого контейнера.

Плагин представляет собой динамическую библиотеку, написанную на языке C#, в которой содержится реализация интерфейса `ISanConnectorPlugin` из библиотеки `CryptoPro.Archive.Plugins.Design`.

Плагин необходимо расположить в папке `cp-archive/plugins` (она расположена рядом с папками `admin-api`, `client-api` и так далее). По умолчанию это `/opt/cp-archive/plugins` на UNIX и `C:\inetpub\cp-archive\plugins` на Windows. Плагин используется программой `signature-updater` — убедитесь, что она установлена.

Создайте C#-проект и в конфигурационном файле `.csproj` укажите следующие настройки:

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <TargetFramework>net6.0</TargetFramework>
        <ImplicitUsings>enable</ImplicitUsings>
        <Nullable>enable</Nullable>
        <RootNamespace>FilesystemSanConnectorPlugin</RootNamespace>
        <EnableDynamicLoading>true</EnableDynamicLoading>
    </PropertyGroup>
    <ItemGroup>
        <Reference Include="CryptoPro.Archive.Plugins.Design">
            <HintPath>path/to/CryptoPro.Archive.Plugins.Design.dll</HintPath>
            <Private>false</Private>
            <ExcludeAssets>runtime</ExcludeAssets>
        </Reference>
    </ItemGroup>
</Project>
```

Важные строки выделены. Замените значение `HintPath` из секции `ItemGroup` на путь к скачанной библиотеке `CryptoPro.Archive.Plugins.Design.dll`. Замените значение `RootNamespace` на название пространства имён плагина, который вы разрабатываете.

Далее реализуйте интерфейс `ISanConnectorPlugin`. Ниже приведён пример плагина, сохраняющего подписи в директорию на жёстком диске и создающего небольшой отчёт о результате усовершенствования.

```
using CryptoPro.Archive.Plugins.Design.Interfaces;
using CryptoPro.Archive.Plugins.Design.Models;

namespace FilesystemSanConnectorPlugin;

/// <summary>
/// Пример плагина, сохраняющего подписи в папку и создающего небольшой отчёт.
/// </summary>
public class FilesystemSanConnectorPlugin : ISanConnectorPlugin
{
    private string _rootFolder;

    /// <inheritdoc/>
    public void Initialize(Dictionary<string, string> parameters)
    {
        var parametersIgnoreCase = new Dictionary<string, string>(
            parameters, StringComparer.InvariantCultureIgnoreCase);

        if (!parametersIgnoreCase.TryGetValue("folder", out _rootFolder))
        {
            _rootFolder = "";
        };
    }

    /// <inheritdoc/>
    public async Task<Result> ProcessAsync(Container container)
    {
        var containerFolder = GenerateContainerFolderName(container.Id);
        if (!Directory.Exists(containerFolder))
        {
            Directory.CreateDirectory(containerFolder);
        }

        var writeSignaturesResult = await WriteSignaturesAsync(
            containerFolder, container.Signatures);
        if (writeSignaturesResult.IsFailure)
        {
            return writeSignaturesResult;
        }

        return await WriteReportAsync(containerFolder, container);
    }

    /// <summary>Записать список подписей в указанную директорию.</summary>
    private async Task<Result> WriteSignaturesAsync(
        string folder, IReadOnlyList<Signature> signatures)
    {
        for (var i = 0; i < signatures.Count; i++)
        {
            var signature = signatures[i];
            var writeSignatureResult = await WriteSignatureAsync(folder, signature, i);
            if (writeSignatureResult.IsFailure)
            {
                return writeSignatureResult;
            }
        }

        return Result.Success();
```

```
}

/// <summary>Записать подпись в указанную директорию.</summary>
private async Task<Result> WriteSignatureAsync(
    string folder, Signature signature, int signatureIndex)
{
    var signaturePath = GenerateSignaturePath(folder, signatureIndex);
    try
    {
        await File.WriteAllBytesAsync(signaturePath, signature.Raw);
    }
    catch (Exception exception)
    {
        return Result.Failure(
            $"Unable to write to file {signaturePath}. Error: {exception.Message}.");
    }

    return Result.Success();
}

/// <summary>Записать отчёт о контейнере в директорию.</summary>
private async Task<Result> WriteReportAsync(string folder, Container container)
{
    var reportPath = GenerateReportPath(folder);

    var clearFileResult = await DeleteFileContentsAsync(reportPath);
    if (clearFileResult.IsFailure)
    {
        return clearFileResult;
    }

    if (container.Status != ContainerStatus.Archived)
    {
        return await WriteReportFailureAsync(reportPath, container);
    }

    return await WriteReportSuccessAsync(reportPath, container);
}

/// <summary>Записать отчёт об успешном усовершенствовании.</summary>
private async Task<Result> WriteReportSuccessAsync(
    string reportPath, Container container)
{
    for (var i = 0; i < container.Signatures.Count; i++)
    {
        var signature = container.Signatures[i];

        var report = GenerateReport(signature, i);
        try
        {
            await File.AppendAllTextAsync(reportPath, report);
        }
        catch (Exception exception)
        {
            return Result.Failure(
                $"Unable to write to file {reportPath}. Error: {exception.Message}.");
        }
    }
```

```
        return Result.Success();
    }

    /// <summary>Записать отчёт о неуспешном усовершенствовании.</summary>
    private async Task<Result> WriteReportFailureAsync(
        string reportPath, Container container)
    {
        try
        {
            await File.AppendAllTextAsync(
                reportPath, $"Error. Container status: {container.Status}.");
        }
        catch (Exception exception)
        {
            return Result.Failure(
                $"Unable to write to file {reportPath}. Error: {exception.Message}.");
        }

        return Result.Success();
    }

    /// <summary>Сгенерировать путь к файлу контейнера.</summary>
    private string GenerateContainerFolderName(Guid containerId) =>
        Path.Combine(_rootFolder, containerId.ToString());

    /// <summary>Сгенерировать путь к файлу подписи.</summary>
    private string GenerateSignaturePath(string folderName, int signatureIndex) =>
        Path.Combine(folderName, $"signature-{signatureIndex}.sig");

    /// <summary>Сгенерировать путь к отчёту.</summary>
    private string GenerateReportPath(string folderName) =>
        Path.Combine(folderName, @"report.txt");

    /// <summary>Сгенерировать отчёт о подписи.</summary>
    private string GenerateReport(Signature signature, int signatureIndex) =>
        $"Signature {signatureIndex} expiration date: {signature.Expires}.\n";

    /// <summary>Удалить текущее содержимое файла.</summary>
    private async Task<Result> DeleteFileContentsAsync(string filepath)
    {
        try
        {
            await File.WriteAllTextAsync(filepath, @"");
        }
        catch (Exception exception)
        {
            return Result.Failure(
                $"Unable to clear file {filepath}. Error: {exception.Message}.");
        }

        return Result.Success();
    }
}
```

Плагин не должен генерировать исключения. Если во время выполнения какой-либо операции произошла ошибка, используйте класс `Result` для передачи информации об этой ошибке в вызывающую функцию.

Также обратите внимание, что при создании плагина будет вызван конструктор без параметров. Для инициализации плагина используйте функцию `void Initialize(Dictionary<string, string> parameters)`. Параметр `parameters` будет прочитан из секции `Plugins` конфигурационного файла `plugin-config.json` из папки с плагинами `cp-archive/plugins`. Если файл отсутствует, создайте его. Пример содержимого конфигурационного файла для плагина выше:

```
{
  "Plugins": {
    "SanConnector": [
      {
        "IsEnabled": true,
        "PluginName": "SanConnector_01",
        "AssemblyName": "FilesystemSanConnectorPlugin.dll",
        "Parameters": {
          "Folder": "/home/username/containers"
        }
      }
    ]
  }
}
```

Секция `Plugins.SanConnector` содержит список, что позволяет подключить несколько плагинов одновременно. Параметр `PluginName` содержит имя плагина, которое используется для его идентификации в дальнейшем. Имя должно быть уникальным и непустым.

При использовании нескольких плагинов все плагины с полем `IsEnabled` со значением `true` будут вызваны по очереди. Очередь может не соответствовать указанной в списке в конфигурационном файле.