



Connect

PLATFORMA DE INTEROPERABILITATE

MConnect Events



Integration guide

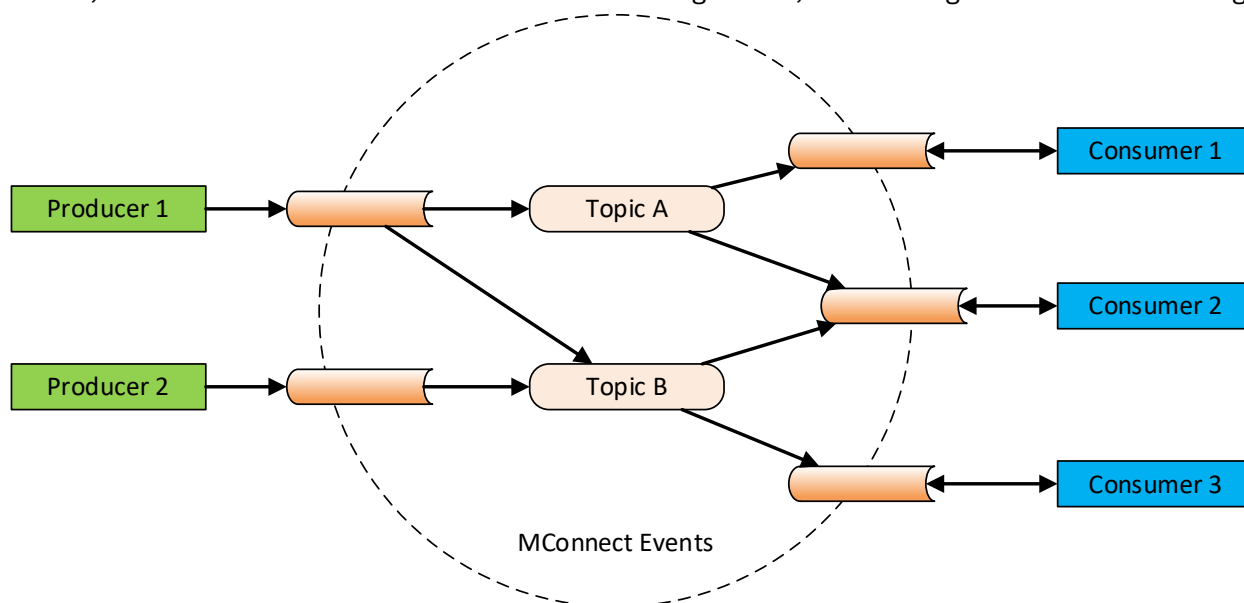
Table of Contents

1	Executive summary	3
2	Glossary of terms.....	4
3	Introduction.....	5
3.1	Scope and target audience	5
3.2	Structure of this document	5
4	Integration checklists.....	6
4.1	Notations	5
5	System context.....	8
5.1	General system capabilities.....	8
5.2	Service dependencies.....	8
5.3	Protocols and standards.....	9
5.4	Limits.....	9
6	Organizational context.....	10
6.1	Service owner.....	10
7	Interaction scenarios.....	11
7.1	Produce events	11
7.2	Consume events	11
8	Integration development	13
8.1	Client credentials and network access.....	13
8.2	Personal data processing.....	13
9	Integration checklists.....	14
10	API Reference.....	16
10.1	Error handling rules	16
10.2	Producer APIs.....	16
10.3	Consumer APIs using WebSocket	19
10.4	Consumer APIs using long polling	20
10.5	Tool APIs	26
11	Samples and Libraries.....	26
11.1	Sample Events and Schemas	26
11.2	Using .NET Integration library	28

1 Executive summary

Data exchange in any Government is a complex endeavor across various levels of effort, including legal, semantic, organizational and technical. In Moldova, the technical level of data exchange is facilitated by MConnect – a national data exchange platform. There are many well-known patterns to implement various data exchange scenarios, including classical request/response messaging, events distribution, large documents distribution and data streaming.

As part of MConnect platform, MConnect Events is the component designed specifically for efficient production and consumption of events. This includes client authentication and authorization as producers and consumers, scalable production and consumption of events, event structure validation when produced, scalable and flexible storage of events awaiting consumption, immediate availability of events to consumers, confirmation of event consumption to ensure reliable delivery of events, as well as internal instruments for configuration, monitoring and troubleshooting.



MConnect Events enables systems to exchange data about various events in real-time as well as in a disconnected way. Events are flowing from producers to MConnect Events then to consumers in their supported pace, whenever they are available. This lowers the coupling between producers and consumers, decreasing their availability and performance requirements.

This document describes the technical interfaces exposed by MConnect Events for client information systems enabling them to produce and consume events. Its target audience is the development teams for those information systems.

The document contains the relevant information required for a complete understanding of MConnect Events from the integration point of view. It contains integration-related technical details, security considerations, as well as describing the integration testing.

Although the document is meant to be technology agnostic, it is also documenting the integration library built for .NET to simplify and speed-up integrations with .NET clients.

2 Glossary of terms

Term	Definition
API	Application Programming Interface
CONNECT	A standard HTTP request method
GET	A standard HTTP request method
HTTP	Hypertext Transfer Protocol
IDNO	Legal entity unique identifier
IDNP	Natural person unique identifier
JSON	JavaScript Object Notation, a format for storing and transporting data
POST	A standard HTTP request method
REST	Representational State Transfer, a software architectural style that defines a set of methods to build web APIs
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URN	Uniform Resource Name
WS	WebSocket, a simultaneous two-way communication protocol working over HTTP
WSS	WebSocket Secure, extension of WS that uses encryption for communication based on TLS

3 Introduction

3.1 Scope and target audience

This document describes the technical interfaces exposed by MConnect Events for client information systems that are using it to produce and consume events. Its target audience is the development teams for those information systems.

3.2 Structure of this document

This document contains the relevant information required for a complete understanding of MConnect Events from the integration point of view. It is also accompanied by samples that exemplify integration scenarios using certain technologies.

The recommended reading sequence are the following chapters:

- **Error! Reference source not found.**
- Interaction scenarios
- Integration development

3.3 Notations

This document contains several notation styles; the following details the styles that have a degree of significance beyond the purpose of communicating information:

Yellow Highlighted Text – Text that is highlighted in yellow irrespective of font attributes (font type, italics, bold, underlined, etc.) means that the text is waiting clarification or verification.

Red Bold Text – Text that is red in color and bold, defines an important piece of information that must be read.

Italic Bold Text – Text that is bold and italic detail actual information or scripts that need to be executed, created, and copied from or to.

4 Integration checklists

Integrations **MUST** be developed and tested within the staging environment only. To ensure high availability, no performance, security or any other kind of tests are allowed on production environment.

The following is a **general checklist** for any client:

- 1) Base address and client certificate are configurable.
- 2) Client certificate private key is secure and differs between staging and production environments.
- 3) Any intermediary certificate is sent with the client certificate during handshake.
- 4) The IP address that is visible to MConnect Events is stable. The address can be a public Internet address or private one from government network.
- 5) Internal procedure is set up to remind system administrators about certificate expiration in advance.

The following is a **checklist for producers**:

- 1) Producers implement an outbox pattern to ensure no events are skipped from being produced.
- 2) Events have a correct URI set in CloudEvent source attribute.
- 3) Events have unique identifiers set in CloudEvent id attribute per source.
- 4) All producer instances from the same source have a consistent value set in CloudEvent time attribute.
- 5) For events that require ordered consumption, the producer sets the partitionkey attribute corresponding to event payload. Partition keys should not be constant as this limits the scalability.
- 6) Events that include personal data include enough information for logging personal data synchronization.
- 7) Each event is not larger than 64 KB.
- 8) Event batches are not larger than 1 MB.

The following is a **checklist for consumers**:

- 1) Events consumption is properly confirmed, either individually or periodically.
- 2) Events are consumed in an idempotent manner, i.e. processing already processed events do not result in double processing or do not create some additional business effect.
- 3) Dead events are properly reported back to MConnect Events.
- 4) Events shall not be considered dead for technical errors, for example due to Consumer network, database or some other component being temporarily inaccessible or in a wrong configuration. Events having a wrong structure are a good example of dead events.
- 5) Consumers use WebSocket protocol.
- 6) Consumers that use long polling for integration use the returned consumer instance base address without any interpretation.
- 7) Consumers that use long polling for integration explicitly delete the instance when shutting down.

- 8) Consumers reconnect when the WebSocket connection is lost or create another long-polling instance when the previous one expires.
- 9) Consumer instances are scaled in divisors of 12.
- 10) Consumers are monitored to run permanently or periodically to not miss events.

4.1 General system capabilities

MConnect Events is a platform-level service, a component of MConnect, the National Interoperability Platform, that allows information systems to efficiently produce and consume events in near real-time.

Our performance tests showed a steady throughput of 10K events/second involving several producers and consumers in parallel. You can produce and consume events from multiple instances of your app using the same client certificate. Additionally, a consumer can repeatedly consume the available events by specifying a different consumer group (instead of using the default one).

MConnect Events authorizes producers and routes events according to each event type. It is possible to authorize multiple producers to produce events of the same event type and configure routing to enable multiple consumers to consume events of the same type. Producer authorization also includes validating each event structure using JSON schema defined for each event type.

Producers can produce a single event or a batch of events. It is important to mention that if one event is wrong, e.g. having unauthorized event type for this producer or invalid structure according to the configured schema, the whole batch is rejected.

In cases that require ordered consumption of events related to a particular real-life entity (such as a person, transaction, etc.), producers must specify a partition key according to CloudEvents partitioning extension.

By default, MConnect Events stores produced events for up to 5 days (120 hours) enabling consumers to consume whenever they are available. To ensure high availability, events are stored in 3 replicas. These settings can be changed upon request when reasonable.

To simplify consumer integration testing, the API allows consumers to produce test events for their own consumption.

Consumers can report back events they are unable to consume (usually due to wrong structure) as dead events. They are stored in a special dead event storage for the consumer and might require further manual intervention.

4.2 Service dependencies

MConnect Events is deployed in a highly available infrastructure and depends only on the availability of MPass API for client management. MPass API is also highly available, however, MConnect Events caches client settings for up to 30 minutes. This ensures high performance of connection opening except for the initial ones, thus decreasing this dependency.

It is also important to note that, due to implementation technicalities, it is normal for consumers to start consuming events with a short delay (several seconds) after opening the consumer connection.

4.3 Protocols and standards

MConnect Events exposes its APIs over HTTPS, supporting HTTP 1.1 and 2. The HTTPS endpoint uses TLS 1.2 and higher and requires authentication through client certificates (encoded in X.509 v3 format).

MConnect Events clients use CloudEvents¹ v1 to produce and consume events using JSON Event Format and HTTP Protocol Binding. Whenever required, producers can specify a partition key, as documented in CloudEvents partitioning extension², while consumers shall use WebSocket Protocol Binding for efficient consumption of events.

The endpoints for producers and long-polling consumers are described in OpenAPI³ 3.0 format. However, for efficient consumers, the recommendation is to use the WebSocket endpoint, which is accessible via the standard HTTP 1.1 Protocol upgrade mechanism⁴, or the standard HTTP 2 CONNECT method (see 8.3 in RFC 7540⁵).

When validating events during production against the configured schema the supported versions of JSON Schema are Draft 6, Draft 7, Draft 2019-09 and Draft 2020-12⁶. The version is identified based on keywords used by the schema.

4.4 Limits

According to CloudEvents standard, it is recommended that the published events be no larger than **64 KB**⁷. This not only ensures any intermediaries will forward the events but also makes the transmission of events efficient and allows broader distribution of events, while stressing the core meaning of events (i.e. messages that inform about something that happened and not a transport for any kind of data). CloudEvents producers SHOULD keep events compact by avoiding embedding large data items into event payloads and rather use the event payload to link to such data items.

All HTTP requests sent to MConnect Events can have up to **1 MB** in size. This means any HTTP message sent by the producer, an events batch or one single event, shall not be larger than 1 MB.

By default, MConnect Events keeps unconsumed events up to **5 days (120 hours)**. This means that if a consumer is inactive for that time, it might miss events.

By default, MConnect Events consumers in a group can be efficiently scaled in **divisors of 12**, e.g. 1, 2, 3, 4, 6 or 12 consumers.

Should any of these limits not cover your practical scenarios, you can request their adjustments having a **proper justification**.

¹ <https://cloudevents.io>

² <https://github.com/cloudevents/spec/blob/v1.0.2/cloudevents/extensions/partitioning.md>

³ <https://www.openapis.org>

⁴ <https://http.dev/protocol-upgrade>

⁵ <https://httpwg.org/specs/rfc7540.html>

⁶ <https://json-schema.org/specification>

⁷ <https://github.com/cloudevents/spec/blob/v1.0.2/cloudevents/spec.md#size-limits>

5 Organizational context

5.1 Service owner

Service Owner	
Organization	e-Government Agency of Moldova
General Point of Contact	
E-mail	office@egov.md
Technical Point of Contact	
Name	MConnect Support and Integrations
E-mail	suport.mconnect@egov.md

6 Interaction scenarios

MConnect Events integrates two types of clients: event producers and event consumers. A client information system can be configured to be either a producer, a consumer, or both.

As defined by CloudEvents standard, all events have a type. In MConnect Events context, types are named using the following convention: “Organization.System.Entity.Action” (for example “AGE.MPass.User.Authenticated”). A Producer is authorized to produce and a Consumer to consume only certain types of events.

6.1 Produce events

A client system that is authorized as Producer can only produce events of **authorized types**. Produced events are authorized, **validated against the configured schema** and persisted in one or more places for consumption. MConnect Events then responds with HTTP 200 OK when all of these succeed, thus ensuring reliable messaging.

A Producer can produce events one by one or in batches. Note that there are some general limits applied to the size of each event and the entire message (see Limits). MConnect Events persists either all events to one or more destination consumers or none, in a transactional manner. This means that it is safe for a Producer to **retry producing events** on errors. Nonetheless, taking into account that it is possible for the error to not get back due to networking or other kind of issues, retries might result in duplicate events produced. To minimize duplication, it is important to assign a unique id attribute for each event as specified by CloudEvents standard and use it including on retry.

Business processes based on event exchange typically require that no event must be skipped, due either to business or technical errors. Thus, to ensure at least once delivery, it is recommended that producers implement the **outbox pattern**⁸ in their informational systems. This will ensure events will be sent or not according to transactional changes in their databases.

A Producer can produce events in parallel, from multiple instances. MConnect Events is a scalable system and can handle a significant number of events.

For performance reasons, MConnect Events does not by default guarantee ordered consumption of events, meaning that events that are close in time might be seen by consumers in a different order they were produced by producers. **If ordered consumption is required** for some entity (for example a person or document), Producer must set partition key to entity identifier according to CloudEvents standard partitioning extension (for example for a person set partitionkey to idnp:{idnp}).

6.2 Consume events

A client system that is authorized as Consumer can only consume events of authorized types. Consumers are actively polling for their events using either HTTP long polling technique or by implementing the efficient and strongly recommended **WebSocket protocol** defined in this

⁸ https://en.wikipedia.org/wiki/Inbox_and_outbox_pattern

document. In both cases, MConnect Events returns pending events for consumption, each of them having an associated offset restarted when consumer connects.

To ensure reliable delivery of events in at-least-once manner, the Consumer must **confirm successful consumption of events**. Each consumed event includes an offset that is restarted at the beginning of consumer session. The consumer confirms the consumption of each event or, for efficiency, a batch of events by including the offset of the last successfully consumed event in the confirmation.

For events that cannot be consumed due to business reasons (required field missing, invalid data, etc.), the Consumer can modify and report the event to MConnect Events as dead. **Dead events** are stored in a special place for manual review or for another instance of Consumer that explicitly consumes dead events. It is important to note that events shall not be considered dead for technical errors, for example due to Consumer network, database or some other component not being available.

Technical errors, non-delivered confirmations and other potential issues might result in the same events being consumed by the Consumer. Thus, it is very important that the Consumer **consumes all events in an idempotent manner**. This means that, depending on the implemented business logic, the Consumer must distinguish initial consumption of event from repeated one, processed partially or completely, then finalize the processing and confirm the consumption similarly in all cases.

A Consumer can consume events in parallel, from **multiple instances**.

For systems that have multiple subcomponents that need to consume the same events, set the **consumer group** parameter to the name of the subcomponent. Test and dead events will be seen and consumed by all consumer instances, irrespective of any indicated consumer group.

7 Integration development

7.1 Client credentials and network access

Before being able to interact with MConnect Events, a client must be registered accordingly by the Service owner. To perform such a registration, please provide your system certificate. If you don't have one, you can request a system certificate for authentication from Information Technology and Cyber Security Service or E-Government Agency.

For security reasons, the client **MUST** use a different certificate for integration with staging and production environments, and corresponding private keys **MUST** be kept as confidential as possible. MConnect Events does not require access to client private keys for integration.

MConnect Events API is accessible only to a registered set of IP addresses and, for security sensitive information systems, this means configuring routes and/or a VPN between the client and MConnect Events.

To register a client and get network access, please write a request by e-mail to the Service owner, providing your public IP address or the VPN assigned private IP address and public key certificate.

7.2 Personal data processing

MConnect Events logs details related to the consumption of events that include personal data. This requires the following details:

- Legal entity identifier (IDNO) – taken from consumer system registration.
- Legal basis for personal data processing – taken from consumer configuration (per source or event type) or extracted from each payload using configured JSON path.
- Legal reason for personal data processing – taken from consumer configuration (per source or event type) or extracted from event payload using configured JSON path.
- Personal data subject – extracted from event payload using configured JSON path

As events are produced and then consumed without a consumer user's explicit request, the personal data processor is considered the consumer's system.

8 Integration checklists

Integrations **MUST** be developed and tested within the staging environment only. To ensure high availability, no performance, security or any other kind of tests are allowed on production environment.

The following is a **general checklist** for any client:

- 6) Base address and client certificate are configurable.
- 7) Client certificate private key is secure and differs between staging and production environments.
- 8) Any intermediary certificate is sent with the client certificate during handshake.
- 9) The IP address that is visible to MConnect Events is stable. The address can be a public Internet address or private one from government network.
- 10) Internal procedure is set up to remind system administrators about certificate expiration in advance.

The following is a **checklist for producers**:

- 9) Producers implement an outbox pattern to ensure no events are skipped from being produced.
- 10) Events have a correct URI set in CloudEvent source attribute.
- 11) Events have unique identifiers set in CloudEvent id attribute per source.
- 12) All producer instances from the same source have a consistent value set in CloudEvent time attribute.
- 13) For events that require ordered consumption, the producer sets the partitionkey attribute corresponding to event payload. Partition keys should not be constant as this limits the scalability.
- 14) Events that include personal data include enough information for logging personal data synchronization.
- 15) Each event is not larger than 64 KB.
- 16) Event batches are not larger than 1 MB.

The following is a **checklist for consumers**:

- 11) Events consumption is properly confirmed, either individually or periodically.
- 12) Events are consumed in an idempotent manner, i.e. processing already processed events do not result in double processing or do not create some additional business effect.
- 13) Dead events are properly reported back to MConnect Events.
- 14) Events shall not be considered dead for technical errors, for example due to Consumer network, database or some other component being temporarily inaccessible or in a wrong configuration. Events having a wrong structure are a good example of dead events.
- 15) Consumers use WebSocket protocol.
- 16) Consumers that use long polling for integration use the returned consumer instance base address without any interpretation.
- 17) Consumers that use long polling for integration explicitly delete the instance when shutting down.

- 18) Consumers reconnect when the WebSocket connection is lost or create another long-polling instance when the previous one expires.
- 19) Consumer instances are scaled in divisors of 12.
- 20) Consumers are monitored to run permanently or periodically to not miss events.

9 API Reference

9.1 Error handling rules

MConnect Events REST APIs can return the following status codes in case of errors:

HTTP Status Code	Description
400 Bad Request	Returned when something is wrong with your request. For example, the request does not include the client certificate or an intermediary, some header is missing, the format is not a valid JSON, etc. For more details, review the content of the response.
401 Unauthorized	Returned on any authorization error. Either the system is not registered as producer or consumer, has wrong authorization configuration, does not have the rights to publish events of the provided event type, or cannot use the indicated source, etc. For more details, review the content of the response.
404 Not Found	The request URL wrong or consumer instance is not found (expired or not on the provided bridge). For more details, review the content of the response.
413 Content Too Large	Returned when the entire HTTP request is larger than specified in Limits.
422 Unprocessable Entity	Returned when the event payload is not valid against the configured event schema. For more details, review the content of the response.
500 Internal Server Error	Unexpected error. Contact the service owner and report the error.

On success, the returned status code is 200, 201, 202 or 204.

9.2 Producer APIs

Producers can produce events using one of the following APIs.

Important! For production scenarios, is recommended to produce events in batches, using the last endpoint. See also Limits.

Endpoint	POST /ce/produce/raw
Description	Produce a single event in raw form in the body of HTTP request. Set the standard Content-Type header to one of the following: <ul style="list-style-type: none">• <i>application/json</i> – the payload is in JSON format (this is most probably the format you intend to use);• <i>application/octet-stream</i> – the payload is binary (only for special cases);• <i>text/plain</i> – the payload is plain text (only for special cases).
Request Parameters	

Location	Parameter	Type	Description
Header	ce-specversion	string*	The version of the CloudEvents specification which the event uses. This enables the interpretation of the context. This MUST always be set to "1.0".
Header	ce-source	uri*	Identifies the context in which an event happened. This MUST be set to the value (or one of the values) allowed in Producer configuration. Producers MUST ensure that source + id is unique for each distinct event.
Header	ce-id	string*	Identifies the event. Producers MUST ensure that source + id is unique for each distinct event.
Header	ce-type	string*	Contains a value describing the type of event related to the originating occurrence. This attribute is used for authorization, routing, observability, etc.
Header	ce-subject	string	This describes the subject of the event in the context of the event producer (identified by source). A consumer will typically consume events emitted by a source, but the source identifier alone might not be sufficient as a qualifier for any specific event if the source context has an internal sub-structure. Optional.
Header	ce-time	date-time	Timestamp of when the occurrence happened. Cannot be set to a future time. Formatted according to RFC 3339. If the time of the occurrence cannot be determined then this attribute MAY be set to some other time (such as the current time) by the CloudEvents producer, however all producers for the same source MUST be consistent in this respect. In other words, either they all use the actual time of the occurrence, or they all use the same algorithm to determine the value used. Optional, defaults to current time.
Header	ce-partitionkey	string	A partition key for the event, specified to ensure consumption ordering between multiple events for the same partitionkey. Optional.
Response: 202 Accepted – returned when the event persisted successfully for all authorized consumers.			

Endpoint	POST /ce/produce/event		
Description	Produce a single event according to CloudEvents standard, meaning the request body must be a valid JSON object. The standard HTTP Content-Type header must be set to <i>application/cloudevents+json</i> .		
Request Parameters			
Location	Parameter	Type	Description
Body	specversion	string*	The version of the CloudEvents specification which the event uses. This enables the interpretation of the context. This must always be set to "1.0".

Body	source	uri*	Identifies the context in which an event happened. This MUST be set to the value (or one of the values) allowed in Producer configuration. Producers MUST ensure that source + id is unique for each distinct event.
Body	id	string*	Identifies the event. Producers MUST ensure that source + id is unique for each distinct event.
Body	type	string*	Contains a value describing the type of event related to the originating occurrence. This attribute is used for authorization, routing, observability, etc.
Body	datacontenttype	string	Content type of data value. This attribute enables data to carry any type of content, whereby format and encoding might differ from that of the chosen event format. Optional, defaults to <i>application/json</i> . Currently only JSON data is supported by MConnect Events for this endpoint.
Body	subject	string	This describes the subject of the event in the context of the event producer (identified by source). A consumer will typically consume events emitted by a source, but the source identifier alone might not be sufficient as a qualifier for any specific event if the source context has an internal sub-structure. Optional.
Body	time	date-time	Timestamp of when the occurrence happened. Cannot be set to a future time. Formatted according to RFC 3339. If the time of the occurrence cannot be determined then this attribute MAY be set to some other time (such as the current time) by the CloudEvents producer, however all producers for the same source MUST be consistent in this respect. In other words, either they all use the actual time of the occurrence, or they all use the same algorithm to determine the value used. Optional, defaults to current time.
Body	partitionkey	string	A partition key for the event, specified to ensure consumption ordering between multiple events for the same partitionkey. Optional.
Body	data	JSON*	The payload of the event in JSON format.
Response: 202 Accepted – returned when the event(s) persisted successfully for all authorized consumers.			

Endpoint	POST /ce/produce/events
Description	Produce a batch of events according to CloudEvents standard, meaning the request body must be a valid JSON array of JSON objects. The standard HTTP Content-Type header must be set to <i>application/cloudevents-batch+json</i> . Each element of the array has the structure described in the previous endpoint.

	<p>This is the recommended way to produce events if you implement the outbox pattern (which is also recommended), in which case you accumulate a list of events to be produced anyway.</p> <p>MConnect Events persists either all events to one or more destination consumers or none, in a transactional manner. This means that it is safe for a Producer to retry producing the batch of events on errors.</p>
--	--

9.3 Consumer APIs using WebSocket

There are two protocols for event consumption. **WebSocket is the recommended one** for efficiency and performance reasons.

The WebSocket endpoint is accessible via the standard HTTP 1.1 Protocol upgrade mechanism and the standard HTTP 2 CONNECT method.

The WebSocket sub-protocol to be used is:

cloudevents.json

The established WebSocket connection is a simultaneous two-way communication channel. The protocol is quite simple.

9.3.1 Messages sent to Consumer

MConnect Events is streaming the events to be consumed to the client as separate messages in JSON format, looking like the following.

First Sample Message:

```
{ "specversion": "1.0", "source": "urn:source", "id": "sample-id-1001", "type":
"Organization.Event.Occurred", "time": "2025...", "offset": "1", "data": { event-
payload-inline-json } }
```

Second Sample Message:

```
{ "specversion": "1.0", "source": "urn:source", "id": "sample-id-1002", "type":
"Organization.Event.Occurred", "time": "2025...", "offset": "2", "data": { event-
payload-inline-json } }
```

And so on.

The meaning of the properties is the following:

Property	Type	Description
specversion	string*	The version of the CloudEvents specification which the event uses, currently always returned as "1.0".
source	uri*	Identifies the context in which an event happened. Producers MUST ensure that source + id is unique for each distinct event.
id	string*	Identifies the event. Producers MUST ensure that source + id is unique for each distinct event.

type	string*	Contains a value describing the type of event related to the originating occurrence.
subject	string	This describes the subject of the event in the context of the event producer (identified by source). A consumer will typically consume events emitted by a source, but the source identifier alone might not be sufficient as a qualifier for any specific event if the source context has an internal sub-structure. Optional.
time	date-time*	Timestamp of when the event happened or when the event was produced. Formatted according to RFC 3339.
partitionkey	string	A partition key for the event, specified to ensure consumption ordering between multiple events for the same partitionkey. Optional.
offset	string*	Event offset for current consumer instance. Used for explicit confirmations.
data	JSON*	The payload of the event in JSON format.

9.3.2 Messages sent to MConnect Events

The client streams back consumption confirmations or dead events.

A confirmation looks like the following:

```
confirm:<<offset>>
```

meaning “confirm:” prefix followed by offset, where *offset* is string (an always increasing integer formatted as string) found from the incoming event. This results in all events up to the specified offset acknowledged as consumed.

Reporting a dead event looks like the following:

```
dead:{ “specversion”: “1.0”, “source”: “urn:source”, “id”: “sample-id-1002”, “type”:
“Organization.Event.Occurred”, “time”: “2025...”, “offset”: “2”, “data”: { event-
payload-inline-json } }
```

meaning “dead:” prefix followed by the dead event JSON, which the consumer might modify if required for later special handling of dead events.

Any other message prefix will result in MConnect Events closing the WebSocket connection.

9.4 Consumer APIs using long polling

There are two protocols for event consumption. WebSocket is the recommended one. However, if you use a framework that doesn’t include a WebSocket client (which is highly doubtful) or if you just want to try event consumption using Swagger UI (or some local HTTP client tool), MConnect Events also implements the well-known long polling protocol.

Long polling requires creating a consumer, polling for events to consume (including sending consumption confirmations) and deleting consumers before closing. Consumers that are not actively polling for events are deleted automatically after some expiration time.

Endpoint	POST /ce/consumers		
Description	Creates a stateful consumer instance on one of the bridges that can be used to consume events in a long polling manner. It is normal for this endpoint to take some time (usually up to 30 seconds), as creating consumers requires some internal coordination.		
Request Parameters			
Location	Parameter	Type	Description
Query	events	boolean	Specifies whether to consume standard events produced by producers. Optional, defaults to true.
Query	test	boolean	Specifies whether to consume test events produced by the calling consumer for testing purposes (see Tool APIs). Optional, defaults to true.
Query	dead	boolean	Specifies whether to consume dead events produced by the calling consumer. Optional, defaults to false.
Query	group	string	Specifies consumer group name. Set by systems that need to consume the events twice in two subcomponents. Do not set this parameter when consuming events in parallel from multiple instances of the same consumer, meaning you don't need to consume the same events multiple times. Optional, defaults to “~default”.
Response: 201 Created			
Location	Parameter	Type	Description
Header	Location	uri*	An absolute URL that is the consumer instance base address for the created consumer instance. Currently it has the following form: https://{mconnect-events-base-address}/{bridge}/ce/consumers/{group}/instances/{instance} having the following path parameters: <i>bridge</i> – the instance of the bridge that the consumer was created on; <i>group</i> – the name of group for the created consumer; <i>instance</i> – consumer instance identifier. Note that the form might be changed in the future, so you MUST use it just as the base address for the other calls related to this instance.

Endpoint	GET /{bridge}/ce/consumers/{group}/instances/{instance}/raw		
Description	Consume the next event as raw, if any. Event payload is returned in the HTTP body.		
Request Parameters			
Location	Parameter	Type	Description
Path	bridge	string*	The instance of the bridge that the consumer was created on. Part of consumer instance base address.

Path	group	string*	The name of the consumer group. Part of consumer instance base address.
Path	instance	string*	Consumer instance identifier. Part of consumer instance base address.
Query	confirm	boolean	Specifies whether to confirm previously consumed events. Optional, defaults to false.
Response: 200 OK			
Location	Parameter	Type	Description
Header	Content-Type	string*	The type of the event payload returned in the body. Can be: <ul style="list-style-type: none"> <i>application/json</i> – the payload is in JSON format (most used format); <i>application/octet-stream</i> – the payload is binary (only for special cases); <i>text/plain</i> – the payload is plain text (only for special cases).
Header	ce-specversion	string*	The version of the CloudEvents specification which the event uses. This enables the interpretation of the context. Always set to “1.0”.
Header	ce-source	uri*	Identifies the context in which an event happened. Producers MUST ensure that source + id is unique for each distinct event.
Header	ce-id	string*	Identifies the event. Producers MUST ensure that source + id is unique for each distinct event.
Header	ce-type	string*	Contains a value describing the type of event related to the originating occurrence.
Header	ce-subject	string	This describes the subject of the event in the context of the event producer (identified by source). A consumer will typically consume events emitted by a source, but the source identifier alone might not be sufficient as a qualifier for any specific event if the source context has an internal sub-structure. Optional.
Header	ce-time	date-time*	Timestamp of when the event happened or when the event was produced. Formatted according to RFC 3339.
Header	ce-partitionkey	string	A partition key for the event, specified to ensure consumption ordering between multiple events for the same partitionkey. Optional.
Header	ce-offset	string*	Event offset for current consumer instance. Used for explicit confirmations.
Response: 204 No Content – returned when there are no events to consume. Returned after poll timeout, during which no producers produced events for the calling consumer.			

Endpoint	GET /{bridge}/ce/consumers/{group}/instances/{instance}/event
-----------------	---

Description	Consume the next event using CloudEvents JSON format.		
Request Parameters			
Location	Parameter	Type	Description
Path	bridge	string*	The instance of the bridge that the consumer was created on. Part of consumer instance base address.
Path	group	string*	The name of the consumer group. Part of consumer instance base address.
Path	instance	string*	Consumer instance identifier. Part of consumer instance base address.
Query	confirm	boolean	Specifies whether to confirm previously consumed events. Optional, defaults to false.
Response: 200 OK			
Location	Parameter	Type	Description
Header	Content-Type	string	The type of the HTTP response content: <i>application/cloudevents+json</i>
Body	specversion	string*	The version of the CloudEvents specification which the event uses, currently always returned as "1.0".
Body	source	uri*	Identifies the context in which an event happened. Producers MUST ensure that source + id is unique for each distinct event.
Body	id	string*	Identifies the event. Producers MUST ensure that source + id is unique for each distinct event.
Body	type	string*	Contains a value describing the type of event related to the originating occurrence.
Body	subject	string	This describes the subject of the event in the context of the event producer (identified by source). A consumer will typically consume events emitted by a source, but the source identifier alone might not be sufficient as a qualifier for any specific event if the source context has an internal sub-structure. Optional.
Body	time	date-time*	Timestamp of when the event happened or when the event was produced. Formatted according to RFC 3339.
Body	partitionkey	string	A partition key for the event, specified to ensure consumption ordering between multiple events for the same partitionkey. Optional.
Body	offset	string*	Event offset for current consumer instance. Used for explicit confirmations.
Body	data	JSON*	The payload of the event in JSON format.
Response: 204 No Content – returned when there are no events to consume. Returned after poll timeout, during which no producers produced events for the calling consumer.			

Endpoint	GET /{bridge}/ce/consumers/{group}/instances/{instance}/events
Description	Consume the next batch of events using CloudEvents JSON format. This method collects a batch of events before returning.

Request Parameters			
Location	Parameter	Type	Description
Path	bridge	string*	The instance of the bridge that the consumer was created on. Part of consumer instance base address.
Path	group	string*	The name of the consumer group. Part of consumer instance base address.
Path	instance	string*	Consumer instance identifier. Part of consumer instance base address.
Query	confirm	boolean	Specifies whether to confirm previously consumed events. Optional, defaults to false.
Response: 200 OK			
Location	Parameter	Type	Description
Header	Content-Type	string	The type of the HTTP response content: <i>application/cloudevents-batch+json</i>
Body	N/A	JSON array	Each element of the array has the structure described in the previous endpoint.
Response: 204 No Content – returned when there are no events to consume. Returned after poll timeout, during which no producers produced events for the calling consumer.			

Endpoint	POST /{bridge}/ce/consumers/{group}/instances/{instance}/confirm		
Description	Confirm the successful consumption of all read events or up to the specified offset.		
Request Parameters			
Location	Parameter	Type	Description
Path	bridge	string*	The instance of the bridge that the consumer was created on. Part of consumer instance base address.
Path	group	string*	The name of the consumer group. Part of consumer instance base address.
Path	instance	string*	Consumer instance identifier. Part of consumer instance base address.
Query	offset	string	Specifies the offset of the last event up to which the consumption of events is confirmed. Optional. When not set, all events read by this consumer instance are confirmed as consumed.
Response: 204 No Content – returned upon successful confirmation.			

Endpoint	POST /{bridge}/ce/consumers/{group}/instances/{instance}/dead		
Description	Produce a dead event for the calling consumer in raw format. Set the standard Content-Type header to one of the following: <ul style="list-style-type: none"> <i>application/json</i> – the payload is in JSON format (this is most probably the format you intend to use); <i>application/octet-stream</i> – the payload is binary (only for special cases); <i>text/plain</i> – the payload is plain text (only for special cases). 		

Request Parameters			
Location	Parameter	Type	Description
Path	bridge	string*	The instance of the bridge that the consumer was created on. Part of consumer instance base address.
Path	group	string*	The name of the consumer group. Part of consumer instance base address.
Path	instance	string*	Consumer instance identifier. Part of consumer instance base address.
Header	ce-specversion	string*	The version of the CloudEvents specification which the event uses. This enables the interpretation of the context. This MUST always be set to "1.0".
Header	ce-source	uri*	Identifies the context in which an event happened.
Header	ce-id	string*	Identifies the event.
Header	ce-type	string*	Contains a value describing the type of event related to the originating occurrence.
Header	ce-subject	string	This describes the subject of the event in the context of the event producer (identified by source). A consumer will typically consume events emitted by a source, but the source identifier alone might not be sufficient as a qualifier for any specific event if the source context has an internal sub-structure. Optional.
Header	ce-time	date-time	Timestamp of when the occurrence happened. Formatted according to RFC 3339. Optional, defaults to current time.
Header	ce-partitionkey	string	A partition key for the event, specified to ensure consumption ordering between multiple events for the same partitionkey. Optional.
Response: 202 Accepted – returned when the dead event persisted successfully.			

Endpoint	DELETE /{bridge}/ce/consumers/{group}/instances/{instance}		
Description	Delete (i.e. close) consumer instance. Shall be called before the consumer is shut down. Calling this explicitly ensures efficient resources usage and faster reconnection of consumer.		
Request Parameters			
Location	Parameter	Type	Description
Path	bridge	string*	The instance of the bridge that the consumer was created on. Part of consumer instance base address.
Path	group	string*	The name of the consumer group. Part of consumer instance base address.
Path	instance	string*	Consumer instance identifier. Part of consumer instance base address.
Response: 204 No Content – returned upon successful deletion of consumer instance.			

9.5 Tool APIs

Tool APIs are intended for human users (meaning developers) for additional information and testing. **Do not call them from your systems.**

Endpoint	GET /ce/tools/my-settings		
Description	Returns the settings configured for the calling client.		
Response: 200 OK			
Location	Parameter	Type	Description
Body	N/A	JSON*	Settings configured for the calling client, according to the internal format that might be changed at any time without prior notice. This is useful for developers to review the configuration for reference and to spot any potential issues.

Endpoint	POST /ce/tools/consumer/test
Description	Enables Consumer developers to produce a test event. Note that, in the case of first call to this endpoint, it is normal for the consumer that is already connected using WebSocket to consume test events after some time (up to 30 minutes), as consumer settings are cached. The structure of the request, response and behavior is similar produce raw event endpoint (see above: POST /ce/produce/raw).

10 Samples and Libraries

10.1 Sample Events and Schemas

Here is a sample CloudEvent in “*application/cloudevents+json*” format:

```
{
  "specversion": "1.0",
  "source": "urn:asp",
  "id": "23fa9243-fe2b-4bdc-8607-ab56d091622b",
  "type": "ASP.RST.VP.Invalidated",
  "time": "2025-03-01T23:09:03.2261311Z",
  "data": {
    "IDNP": "2134567890123",
    "Number": "123456789"
  }
}
```

The event above represents the fact that VP (Vehicle Registration Certificate) with Number “123456789” was invalidated and the owner of the vehicle has “2134567890123” as IDNP. The schema for the above event is the following:

```
{
  "type": "object",
```

```

    "properties": {
      "IDNP": {
        "type": "string",
        "pattern": "^\\d{13}$"
      },
      "Number": {
        "type": "string"
      }
    },
    "required": [ "IDNP", "Number" ],
    "additionalProperties": false
  }

```

Here is another CloudEvent example that is considered as personal data processing:

```

{
  "specversion": "1.0",
  "source": "urn:asp",
  "id": "3dfce534-cdc9-44d5-ba39-690f69706589",
  "type": "MS.eCMND.Person.Born",
  "time": "2025-03-02T09:30:02+02:00",
  "data": {
    "ChildIDNP": "2134567890123",
    "MotherIDNP": "2134567890122",
    "HospitalID": 37,
    "CaseID": 10327,
    "EventReason": "Sincronizare date naștere 10327"
  }
}

```

The event above represents the fact of birth and includes EventReason that can be used as legal reason for personal data processing. In this case, MConnect Events is configured to extract the legal reason from EventReason field (using "\$.EventReason" as JSON path).

The schema for the above event is the following:

```

{
  "type": "object",
  "properties": {
    "ChildIDNP": {
      "type": "string",
      "pattern": "^\\d{13}$"
    },
    "MotherIDNP": {
      "type": "string",
      "pattern": "^\\d{13}$"
    },
    "HospitalID": {
      "type": "integer"
    },
    "CaseID": {

```

```

        "type": "integer"
    },
    "EventReason": {
        "type": "string"
    }
},
"required": [ "ChildIDNP", "MotherIDNP", "HospitalID", "CaseID", "EventReason" ],
"additionalProperties": false
}

```

10.2 Using .NET Integration library

For .NET clients, e-Government Agency developed an integration library, named **Age.Integrations.MConnect.Events**, available as a NuGet package either from the internal artifact Feeds or upon request.

10.2.1 Configuring system certificate

To configure a producer or consumer, the client must first ensure the system certificate is added (as used for all platform-level services integration). The following code does that:

```

builder.Services.AddSystemCertificate(builder.Configuration.GetSection("Certificate")
);

```

The above code expects the following configuration section (in appsetting.json or from other configuration sources):

```

"Certificate": {
    "Path": "path to pfx file or mounted Kubernetes secret as folder",
    "Password": "password for pfx file"
}

```

10.2.2 Producing events

Then the client can configure a producer:

```

builder.Services.AddCloudEventsProducer(builder.Configuration.GetSection("CloudEvents
Producer"));

```

with the following configuration section:

```

"CloudEventsProducer": {
    "BaseAddress": "https://mconnect-events.staging.egov.md:8443/ce/"
}

```

Here is the comprehensive list of configuration keys for a producer:

- *BaseAddress*: The base address for MConnect Events endpoint. Must be explicitly set.
- *Timeout*: Timeout for produce calls. Defaults to 100 seconds.
- *JsonSerializerOptions*: Serializer options to use when serializing CloudEvent data to JSON.

The resulting service, ***ICloudEventsProducer***, available from .NET Core dependency injection container, includes several overloaded methods named ***ProduceAsync*** that allow producing single instances of *CloudEvent* or a list of them as a batch.

CloudEvent.Id shall be unique for all events and *CloudEvent.Source* shall be set as a valid URN set in producer configuration.

Note that to ensure ordered consumption of events related to particular entity, set the *CloudEvent.PartitionKey* to the same value, such as entity identifier with a prefix (e.g. “idno:1010600034203”).

10.2.3 Consuming events

To configure a consumer, call:

```
builder.Services.AddCloudEventHandlers(builder.Configuration.GetSection("CloudEventsConsumer"))
```

with the following configuration section:

```
"CloudEventsConsumer": {  
  "BaseAddress": "wss://mconnect-events.staging.egov.md:8443/ce/"  
}
```

Here is the comprehensive list of configuration keys for a consumer:

- *BaseAddress*: The base address for MConnect Events web-socket endpoint. Must be explicitly set.
- *ConnectTimeout*: Timeout for connection opening. Defaults to 30 seconds.
- *ReceiveBufferSize*: Buffer size to receive data in bytes. Defaults to 64 * 1024 bytes (64 KB).
- *ConsumeEvents*: Specifies whether standard events must be consumed. Defaults to true.
- *ConsumeTest*: Specifies whether test events must be consumed. Defaults to true.
- *ConsumeDead*: Specifies whether dead events must be consumed. Defaults to false.
- *Group*: The group this consumer belongs to. Defaults to null, meaning a default consumer group. Set this only when you want to consume the same events in a different consumer group.

Then fluently add one or more handlers, using either ***AddSingletonHandler<THandler, TData>*** or ***AddTransientHandler<THandler, TData>*** methods, where *THandler* implements ***IHandleCloudEvents<TData>*** interface, and *TData* is a strongly typed event data. You can also control the deserialization by providing an instance of *JsonSerializerOptions* to *AddXXXHandler* methods.

Alternatively, if you need custom logic for event handler identification and data deserialization, you can add a generic implementation of ***ICloudEventsConsumer*** by calling ***AddCloudEventsConsumer<TConsumer>***. That handler will receive all events that the consumer can consume.

In both cases, the implementations of ***IHandleCloudEvents.HandleAsync*** and ***ICloudEventsConsumer.ConsumeAsync*** shall call *ConfirmAsync* on the provided context. In cases

where the received event cannot be consumed, the consumers can call `DeadAsync` on the provided context, to report the event as dead. Dead events require manual intervention of MConnect Event administrators or the consumer calling with `ConsumeDead` set to true in configuration.



Important!

While this guide outlines the general steps to integrate with the MConnect platform, please note that certain integration parameters are not publicly accessible for security reasons. To proceed with full integration, you must request access to the necessary integration data. Access is granted upon request and validation.

For this, please contact the MConnect Support Team using the details below:

Email: suport.mconnect@egov.md

Phone: +373 22 820 022

