HTTP Header Support

The xUML Runtime comes with many adapters (see xUML Service Adapters) that allow you to access a variety of backends via different interfaces. Request to REST, SOAP and HTTP services, and via REST, SOAP and URL adapters use the HTTP protocol. With this protocol, the xUML Runtime serves a set of standard headers (via **libcurl**) as described below (Standard HTTP Headers with xUML Service Adapters). You can overwrite these standard headers using the concept of **HTTP header roles**.

Also, the Runtime provides interfaces to act as a variety of services. In general, the Runtime supports HTTP 1.0 for services but some features of HTTP 1.1 are implemented as well (see Standard HTTP Headers with Service Implementations below).

On this Page:

Related Pages:

- E2E Adapters
- Service Implementations
  - HTTP Service
  - REST Service
  - SOAP Service
- Frontend Components

Standard HTTP Headers with xUML Service Adapters

SOAP, REST and URL adapters use **libcurl** to provide HTTP headers.

Runtime 2019.9 With xUML service adapter calls, the xUML Runtime adds the following outgoing HTTP headers containing correlation information to the request:

- **X-Transaction-Id** or **xTransactionId** (in JMS context)
  This header identifies the transaction the call belongs to. You can set the transaction id manually with setTransactionID. If not set, the Runtime will generate one.
  This header will be passed through the callstack to identify all service calls that belong to a transaction.
- **X-Request-Id**
  This header identifies the unique request. The Runtime generates a unique number for each adapter call.
- **X-Sender-Host** and **X-Sender-Service**
  These headers contain the sender host resp. the sender service. They are set by the Runtime automatically.

Transaction id and request id will be logged to the transaction log on the adapter call. Having this information, you can use this for error analysis or usage metrics.

For more information on specific adapters refer to xUML Service Adapters.

Standard HTTP Headers with Service Implementations

In general, the Bridge supports HTTP version 1.0 for xUML services. However, the following features of HTTP 1.1 are implemented to the xUML Runtime as well:

- `Expect: 100 Continue`
- Runtime 2018.5 `Transfer-Encoding: chunked`

- Runtime 2019.9 Bridge xUML services read the following incoming HTTP headers containing correlation information:
  - **X-Transaction-Id** or **xTransactionId** (in JMS context)
    This header identifies the transaction the call belongs to. You can set the transaction id manually with setTransactionID. If not set, the Runtime will generate one.
    This header will be passed through the callstack to identify all service calls that belong to a transaction.
  - **X-Request-Id**
    This header should identify the unique request.
  - **X-Sender-Host** and **X-Sender-Service**
    These headers should contain the sender host resp. the sender service.

  These headers will be all logged to the transaction log. Having this information, you can use this for error analysis or usage metrics.

For more information on specific service implementations refer to Service Implementations.

Overwriting the Standard HTTP Headers

Runtime 2020.12 The standard header handling of the xUML Runtime as described above works well in homogeneous xUML environments. If you need to access services that are implemented using other technologies (e.g. in a distributed environment), however, the standard HTTP header handling may not meet your needs. You may need to e.g. provide correlation IDs in a different header, or overwrite standard header names.
With the concept of **HTTP Header Roles**, you can take control of the header handling by defining your own header roles.

> Once you define header roles, the standard header handling will be completely dropped and replaced by your configuration.

Once header roles are defined, they are changeable as settings on the Bridge (see xUML Service Settings).

Configuring Incoming Header Roles on HTTP Services

For all services that are accessible via the HTTP protocol (HTTP, REST, SOAP), you can define your own pairs of `<http header name>:<role>` in the component diagram in tagged value **httpHeaderRoles** on the composite (<<E2EComposite>>). **httpHeaderRoles** can hold a list of definitions in format `<http header name>:<role>` (one list entry per line). See Frontend Components for more details on component diagrams.

This can be used to overwrite the default behavior for HTTP services described above.

Available roles are:

Role	Description
none	Remove the role from a standard header with e.g. `X-Transaction-Id:none`.
client_host	The header identified by <http header name> should be treated as **X-Sender-Host**.
client_service	The header identified by <http header name> should be treated as **X-Sender-Service**.
correlation_id	The header identified by <http header name> should be treated as **X-Request-Id**.
transaction_id	The header identified by <http header name> should be treated as **X-Transaction-Id**.

The following rules apply to the usage of header roles:

- The default header handling is completely dropped and replaced by your definitions (see example 2).
  If you want to use the standard configuration and just exchange one header configuration, you must reconfigure the other standard headers.
- You can assign multiple roles to the same header by listing the header twice in the role definition (see example 5).
- If you apply the same role to different headers, the first one (in order of definition) is used (see example 4).

Some examples:

Nr	Role Definition on Service Composite	Incoming Headers	Headers Regarded by Runtime	Rules Applied
1		X-Sender-Host X-Sender-Service X-Request-Id X-Transaction-Id	X-Sender-Host X-Sender-Service X-Request-Id X-Transaction-Id	• standard header handling
2	**myHeader=client_host**	X-Sender-Host X-Sender-Service X-Request-Id X-Transaction-Id		• role definition overwrites standard handling, standard headers are disregarded

		myHeader X-Sender-Host X-Sender-Service X-Request-Id X-Transaction-Id	**myHeader**	• role definition overwrites standard handling, standard headers are disregarded • incoming user-defined header is regarded
3	**myHeader=client_host** X-Sender-Service=client_service X-Request-Id=correlation_id X-Transaction-Id=transaction_id	X-Sender-Host X-Sender-Service X-Request-Id X-Transaction-Id	X-Sender-Service X-Request-Id X-Transaction-Id	• role definition overwrites standard handling • only the redefined standard headers are regarded
		myHeader X-Sender-Host X-Sender-Service X-Request-Id X-Transaction-Id	**myHeader** X-Sender-Service X-Request-Id X-Transaction-Id	• role definition overwrites standard handling • redefined standard headers are regarded • incoming user-defined header is regarded
4	**myHeader2**=correlation_id **myHeader1**=correlation_id	myHeader1 myHeader2	**myHeader2**	• **myHeader2** takes precedence over **myHeader1** because it comes first in order of definition
		myHeader1	**myHeader1**	• **myHeader1** is regarded because **myHeader2** has not been provided
5	myHeader=**correlation_id** myHeader=**transaction_id**	myHeader	myHeader	• contents of **myHeader** are treated as correlation and transaction id

Incoming Header Roles on REST Services

Additionally, header roles of REST services can be overwritten in the <<E2ERESTService>> definition (same rules as explained above).

> When specifying roles on both, the composite and the REST service, please note that header role definitions on the REST service (<<E2ERESTService>>) overrule the same role definitions on the composite(<<E2EComposite>>).

Some examples:

> **Known Issues**
>
> Example 3 does not behave as documented here. The xUML Runtime will prefer myHeader1 to myHeader2 if both headers are provided. This issue will be fixed with one of the next Runtime releases.

Nr	Role Definition on Service Composite	Role Definition on REST Service	Incoming Headers	Headers Regarded by Runtime	
1			X-Sender-Host X-Sender-Service X-Request-Id X-Transaction-Id	X-Sender-Host	sender host
				X-Sender-Service	sender service
				X-Request-Id	request /correlation id
				X-Transaction-Id	transaction id

2	**myHeader=client_host**		**myHeader** X-Sender-Host X-Sender-Service X-Request-Id X-Transaction-Id	**myHeader**	sender host
3	myHeader1=client_host	**myHeader2=client_host**	myHeader1 myHeader2	**myHeader2**	sender host
			myHeader1	myHeader1	sender host
4	myHeader1=client_host myHeader2=client_service	**myHeader2=client_host**	myHeader1 myHeader2	**myHeader2**	**sender host**
5	myHeader1=client_host myHeader2=client_service	myHeader2=none	myHeader1 myHeader2	myHeader1	sender host
6	myHeader1=client_host	myHeader2=client_service	myHeader1 myHeader2	myHeader2	sender service
				myHeader1	sender host
7	**myHeader2=correlation_id** myHeader1=correlation_id		myHeader1 myHeader2	**myHeader2**	request /correlation id
8	myHeader2=correlation_id myHeader1=correlation_id	**myHeader1=correlation_id**	myHeader1 myHeader2	**myHeader1**	request /correlation id