|  |
| --- |
| CIN/NIC |
| Service Catalogue  Commons |
| NIP-PIN common (standards) elements. |
|  |
| **Atos** |
| **31/01/2012** |

|  |
| --- |
| Description of the common elements of the different business interfaces that NIP-PIN provides via the SSO/ESB or SSO-Only service. Only the business part, or the SOAP-bodies, are describe in this document. |

Contents

[1 Revision table 3](#_Toc347407291)

[2 Introduction 4](#_Toc347407292)

[2.1 Audience 4](#_Toc347407293)

[2.2 Goal of the document 4](#_Toc347407294)

[2.3 Document License 4](#_Toc347407295)

[3 Common for all Services 5](#_Toc347407296)

[3.1 Basic Principal 5](#_Toc347407297)

[3.2 Service versioning 5](#_Toc347407298)

[3.3 Service lifecycle 5](#_Toc347407299)

[3.4 Message Traceability 5](#_Toc347407300)

[3.4.1 With response 5](#_Toc347407301)

[3.4.2 Without response 6](#_Toc347407302)

[3.5 Message Order 6](#_Toc347407303)

[3.6 Interoperability 7](#_Toc347407304)

[3.7 Best Practices 7](#_Toc347407305)

[3.7.1 eHealth SOA Naming conventions 7](#_Toc347407306)

[3.7.2 Simple generated code 7](#_Toc347407307)

[3.7.3 Avoid redundant input 7](#_Toc347407308)

[3.8 Common Input & Output 7](#_Toc347407309)

[3.8.1 Goal 7](#_Toc347407310)

[3.8.2 Format and Usage 7](#_Toc347407311)

[3.8.3 Message Description 8](#_Toc347407312)

[3.8.4 Message variants 24](#_Toc347407422)

# Revision table

|  |  |  |
| --- | --- | --- |
| **Release No.** | **Date** | **Revision Description** |
| 1.0 |  | Previous version |
| 2.0 | 31-01-2013 | Reworked probative force: clarified some point and added second variant on request of eHealth. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Introduction

## Audience

This document is intended for analysts, architect and developers of all applications that interface with the NIP-PIN platform. It includes both Package Providers and Relaying Parties.

## Goal of the document

This document intends to describe the common elements of the various services of NIP-PIN. It doesn’t describe the communication (SSO/ESB or SSO-Only). It provides information about the design of common parts of the various services. This allows architects of client applications to align their own architecture.

This document is referenced from the service-specific descriptions (i.e. the various Service Catalogs).

## Document License

[[TODO]]

# Common for all Services

## Basic Principal

All services will be business specific. This means single service will only provide the functionality for one business transaction. Also it means that the service will only require business information and not technical information.

In some case there might exist several technical variants of the same service (e.g. version, format…). Each variant will be a different method with a different namespace and/or root-element; there will never be a version number in message itself.

## Service versioning

When a service evolves, a new operation is added with a new operation name, a new element-name and/or a new element-namespace. For each of the (existing) operations the lifecycle status will be re‑evaluated on an individual base.

For NIP-PIN versions are more than an evolution of the format of the same message. It also means different formats of the same information. For example, the insurability input/output are different for nurses and pharmacies. Therefore both will have their own version of the input en output message.

This results in 2 unique URL, both WSDL and service itself, for a single business service. These URL’s do not change due to updated while it is still possible for both client (you) and server (us) to determine which version you use.

## Service lifecycle

A service and each of the operations have one of the following lifetime statuses.

* **Maintained**: is active and can be continued to be used
* **Deprecated**: is still active might disappear and/or is no longer maintained. It is advised to migrate to a maintained version of this service.
* **Removed**: is no longer defined in the WSDL and can no longer be used. It is still documented to facilitate the migration of clients.
* **Archived**: no longer defined in both WSDL and documentation.

A service or service operation can change at any moment, but will never be without a cause. Possible causes include changes in the business process, technology evolutions, changes in eHealth, …

## Message Traceability

Message traceability is very important for NIP-PIN. We are capable of tracing the messages from care provider to insurance organization if you provide us with the required information. This information is different when you get a response or not.

### With response

The following references should be provided:

* WS-Addressing message reference (if available for the service)
* Input reference
* Output reference (if provided)
* NIP-PIN reference

Both NIP-PIN and the insurance organizations can use this reference number to find information about the message. Depending on the message it can also be possible to track the message itself. The only additional information needed is the environment (URL) the message was sent too.

Although not 100% necessary it does speed up the process if you provide the Service Desk with the same information as is required for the case where there is no response.

### Without response

A response means a business response while the generic fault is considered without response.

#### With event ID (eHealth ESB only)

With this event ID we can find the reference number and come in the same situation as with response.

#### Without event ID

Normally you will receive a response, even when there is an error, but in some exception cases this might be impossible. In this case it is important to provide as much information as possible. This information can include:

* Service
* Service operation
* Common Input data
* Date/Time
* Fault received (complete, not just message)
* …

## Message Order

For Sync messages the message order is guaranteed for messages that are send sequentially: the next message is only send after the response of the previous message is received. The order of sync messages that are sent before the previous message is response of the previous message is received is undetermined.

For ASync messages the order isn’t guaranteed by default. Once an acknowledge is received, the processing is based on priority, load, message size, randomness and not on the order it is received.

Some Async services are an exception on the rule and keep the order with the during Async processing. The order is determined on the order of reception, but as with Sync the order is only guaranteed for sequentially sent messages. To avoid delay of messages, the order is only kept for messages of the same or related types that originate from the same entity and are for the same receiver. So messages of a different type, from a different hospital or to a different IO will not block each other.

## Interoperability

The goal of the services is to be compatible with as many client types as possible. For this we tend to follow the WS-I Basic Profile.

## Best Practices

The following best practices are followed.

### eHealth SOA Naming conventions

Since this is a Belgian Health Care project we follow the eHealth defined SOA naming conventions. These follow the standard best practices.

### Simple generated code

XSD, the schema language used by NIP-PIN for the web service message format, is very powerful. Although most frameworks are capable of handling any XSD schema it quickly results in generated code that is complex and difficult to use.

NIP-PIN makes sure that the XSD schemas it publishes are targeted for code generation and not so much for message validation. This means we avoid the usage of enums, abstract types, choices, attributes and mostly work with sequences and optional or required elements.

### Avoid redundant input

All service interfaces are designed to avoid multiple occurrences of the same value. For example, it will not be necessary to provide the test flag multiple times.

This principal isn’t executed into the extreme. It is still possible that the same value must be provided 2 times (e.g. NIHII) when there are cases these are different. The common input provides the means of avoiding this, see “Message Variants”.

NIP-PIN will also not request information that it is required but can always deduct from other input. We do always require the information if the information can sometimes be deducted and sometimes not, even in the case where it can be deducted. This is to allow for consistency. This principal does not apply on optional information.

## Common Input & Output

### Goal

Although the interface is completely adapted to the business interface, there is still information that is required/provided with every business transaction. This is mainly because most business transactions NIP-PIN provides are between Care Providers and Insurance Organizations.

The common input and output only applies to information that is common for all NIP-PIN services. Information that is only applicable to some business transactions are not part of the common input and output.

### Format and Usage

The common schema defines the common input and output. One specific version of the common schema is used by a business schema. The common schema has its own namespace and separate version number. A specific version of the business schema references a specific version of the common schema, which never changes. If a service requires a new version of the common schema, a new version of the business schema will be created for this.

### Message Description

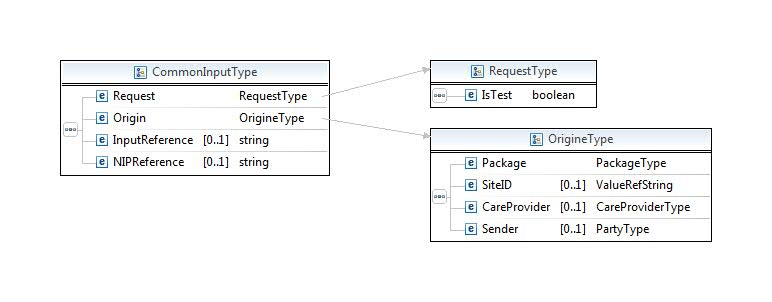
This section describes the schema of both the common input and common output.

#### Input

The common input is always the first child-element of the input-element and has the “CommonInputType”. This contains the information that applies to the all request records.

A common record input is always the ‘preceding-sibling’ or each record. A record is an individual request (or response). In some services may group several individual requests into a single request but most service operations only support a single individual request and then the common record input is right behind the common input.

##### Root Type: Common Input Type



The root type contains the info about the request, the origin and references. The request indicates the type of request, currently only debug or not. InputReference is a reference filled by the requester (the care provider). NIPReference is a reference filled by the NIP-PIN platform and should therefore never be filled in. These references are features of the message as a whole; a message may contain many records. These references are free text value (before v2.3 this was limited to 14 characters), they will be returned in the common record response. When omitted, the value will also be omitted from the response.

The origin indicates where the message originates from and consists of:

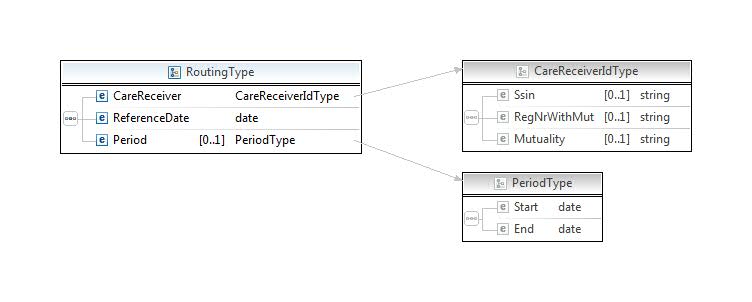
* Package: What software package did send it.
* SiteID (since v2.1): The ID of the site that makes the request. See service catalogue for the allowed values per sector.
* Care Provider: Who was the beneficiary of the request (this is optional in the XSD but should almost always be provided).
* Sender (since v2.2): Who was the creator of the request (in most case this isn’t required)

For those flows that have a flat file equivalent, this corresponds to the segment 100:

* 100: specific for CareNet, is replaced by the “To” of WS-Addressing
* 101: specific for CareNet, is replaced by the “From” of WS-Addressing
* 102: Not part of common input, replaced by “To” of WS-Addressing
* 103: Replaced by “Origin”-element (actual value in //CommonInput/Origin/CareProvider/Nihii/Value)
* 105/106: Not part of common input but of XAdES when relevant
* 107: InputReference, First char repeated in Request/IsTest

##### Routing

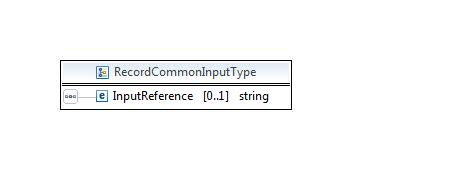
Routing is a type used besides common input and is added in version 2.3 of the interface. It allows NIP-PIN to determine the HIO in case it isn’t defined explicitly in the WS-Addressing “To” element.



The routing consists of the following elements:

* **Care Receiver**: the member of the HIO, it is to this HIO the message is routed too.
  + **SSIN**: the national number of the care receiver, should always be present and can only be omitted in very rare case (e.g. new born).
  + **RegNrWithMut**: the registration number of the care receiver, optional when the SSIN is provided. Must be combined with the mutuality since the number is only unique with a mutuality.
  + **Mutuality**: the 3 digit code of the mutuality (e.g. 203, 100, …). Must be used with reg nr with mut, but can also be used with SSIN to override the mutuality the message should arrive.
* **Reference Date**: The date for which the membership should be resolve, this must be a date within the period of the business request. See the relevant service catalogue for the specific rule.
* **Period**: If the business request has a fixed period length, this can be omitted. In case the business request has a variable period it must be provided. See the relevant service catalogue for the specific rules.

##### Record Root Type: Record Common Input Type

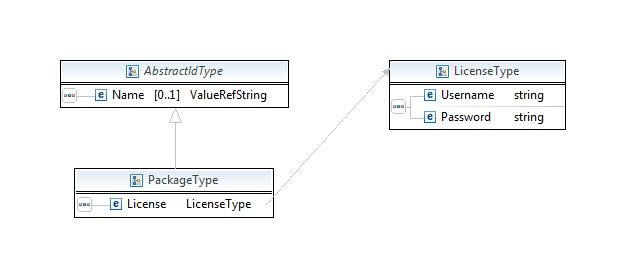


The record root type consists of a reference (before version 2.2 this was limited to a maximum 14 numeric digits) that will be part of the response.

For those flows that have a flat file equivalent, this corresponds to the segment 200:

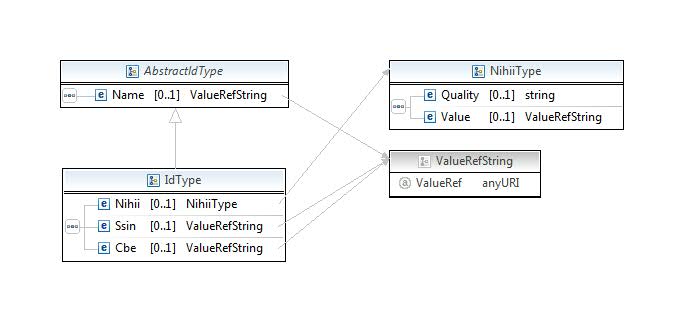
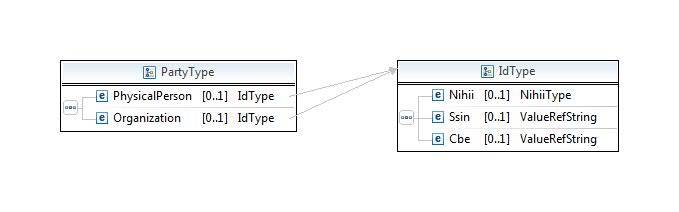
* 200: specific for Flat, XML is “typed” and therefore this value is redundant
* 201: Version is part of the namespace, offering an enforced version.
* 202:
  + Test/Production: part of common input, only 1 field applies to all records.
  + Sync/Async: not repeated for flexibility (e.g. send sync messages Async in case of temporally downtime).
* 203: Replaced by Fault, see below
* 204: Input Reference
* 205: Not applicable here

###### Package Type



A package is identified by its name (optional) and its license. The license is a username password that is distributed to package providers that are licensed to access NIP-PIN.

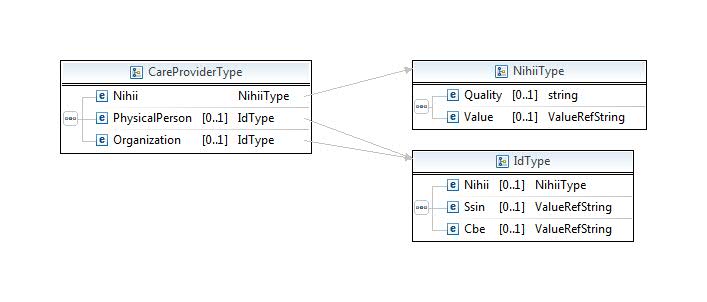
###### Party Type



This type is used to identify a legal entity. This is either: a physical person, an organization or a physical person at an organization.

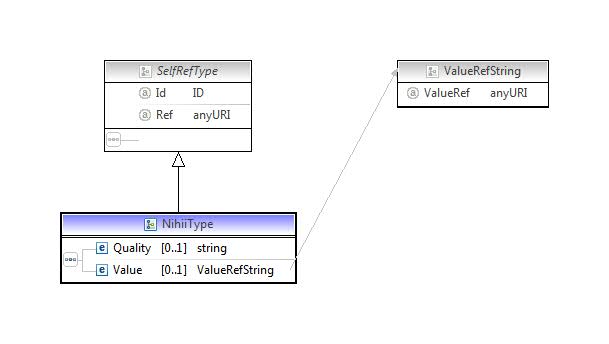
A legal entity is identified by a name (optional) and a unique number. Currently this can be one of the following: SSIN (physical person only), NIHII (health institute only) or CBE (enterprise only).

###### Care Provider Type



A Care Provider is similar to a Party-Type with the extension of a NIHII. The NIHII indicates the beneficiary care provider as know by the INAMI/RIZIV. The physical person and organization elements provide extra information about this care provider.

##### NIHII Type

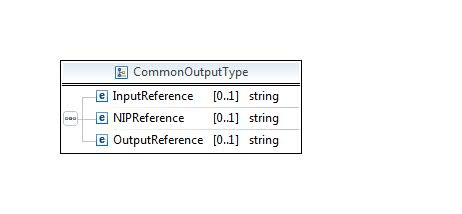


The NIHII consists of the quality (nurse, hospital, pharmacy, …) and the identification number. The different values for the quality are described in the referenced document: “MyCareNet Authentication Catalogue.docx”. See the section “Message variants” for more information about the id, ref and valueRef attributes.

#### Output

As for the input, there is a common output and a common record output. The common output if for the entire message and the common record output is just before every response record.

##### Common Output Type



The InputReference will be empty in the case of common output for Asynchrone.

The NIPReference is a reference generated by NIP-PIN to identify the response message as a whole; this reference is also known by the IO and is linked to the request (cfr NIPReference in CommonInputType). This reference will always be provided, except for some rare cases (e.g. the generation of the reference fails).

The OutputReference is a reference generated by the IO to identify the response as a whole; this reference is only known by the IO (NIP-PIN does not keep a trace of it). This reference is only returned when there was a response from the IO, otherwise this reference is omitted.

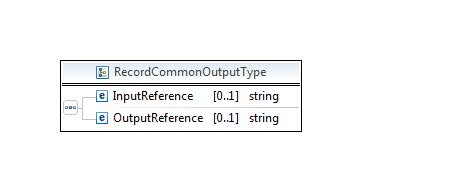
Before version 2.2, all these references where limited to a 14 character long alpha numeric value.

For those flows that have a flat file equivalent, this corresponds to the segment 100:

* 100: specific for CareNet, no replacement.
* 101: specific for CareNet, no replacement.
* 102: Not part of common output, when relevant must be part of the business (which allows multiple values)
* 103: Not part of common output, when relevant must be part of the business (which allows multiple values)
* 105/106: Not part of common input but of XAdES.
* 107: Output reference.

NIP-Reference is new because of the new architecture. Input Reference repeats the reference of the input, but isn’t always available.

##### Common Record Output Type



The InputReference on record level can either be provided by the client (via the record common input) or be generated by NIP-PIN when not provided in the input. Normally this reference is always returned, except in occasional situations (e.g. generation of reference failed).

The OutputReference on record level is always generated by the IO and can be used with the IO to track the response record. It will only be present when there was a response of the IO, otherwise it will be omitted.

This InputReference is omitted in the output when it wasn’t present in the input. The value can be anything and is only intended for client use, NIP-PIN nor the IO can use this to track a message.

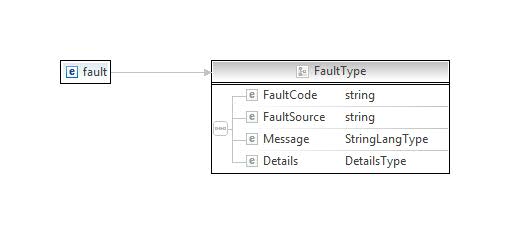
For those flows that have a flat file equivalent, this corresponds to the segment 200:

* 200: specific for Flat, XML is “typed” and therefore this value is redundant
* 201: Version is part of the namespace, offering an enforced version.
* 202:
  + Test/Production: part of common input, only 1 field applies to all records.
  + Sync/Async: not repeated for flexibility (e.g. send sync messages Async in case of temporally downtime).
* 203: Replaced by Fault, see below
* 204: Input Reference
* 205: Output Reference

#### Fault

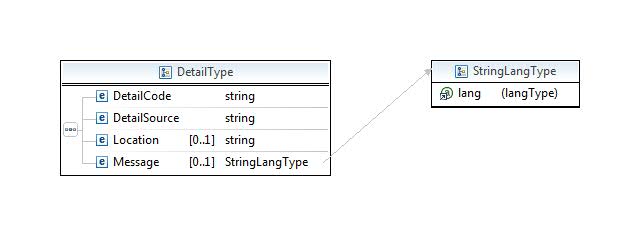
In case a service uses faults, it uses the same fault type (but not fault message). This fault type is adapted to handle all type of technical errors, but not necessary message faults.

The fault has the following structure:



* Fault Code: A human readable (no number) but computer friendly code of the type of fault.
* Fault Source: partner/part that detected the error, important to determine who can assist in the resolution of the error
* Message
  + @Lang: the language of the message (generally “en”)
  + (value): description of the fault type, should be combined with the detail description.
* Details: list of detail messages

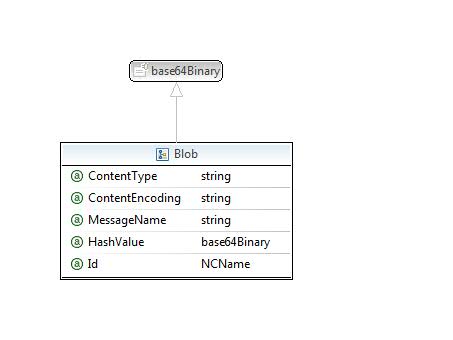
The detail message consists of:



* Detail Code: a human readable code (not a number) of the error that was detected.
* Location: stack-trace extract, xpath or description of the location where the error occurred.
* Message
  + @Language: The language of the detail message (generally “en”)
  + (value): The detailed message.

#### Blob

This document introduces a new type: blob. This is used to transfer (large) binary objects. It has the following structure:



* ***(value)***: The binary data, first compressed via deflate and then base 64 encoded (compatible with MTOM/XOP).
* ***@ContentType*** (required): exactly like xmine:contentType, but in a different namespace to avoid conflicts with MTOM/XOP. It must define which type the content is, it must be values like “text/plain”, “text/xml”, “application/pdf”, …
* ***@ContentEncoding***: a fixed value “deflate” to indicate that the content is compressed via deflates.
* ***@MessageName*** (required): The business name of the message, e.g. “FAC”.
* ***@HashValue:*** pre-calculated hash of the uncompressed and decoded content. Must not be provided by the care provider, always provided to the care provider.
* ***@Id***: The ID of the blob for usage in the XAdES signature. It is an “NCName” instead of an “ID” in order to be able to have different blobs with the same (fixed) id without causing an XSD validation.

#### XAdES

When probative force is required, MyCareNet uses XAdES by default to obtain this. It is part of the request and/or response and identifies the origin, proofs the data and protects against modifications. It does not protect against unauthorized reads.

XAdES is a standard by ETSI that extends the XML Digital Signature to conform to the rule of "Directive 1999/93/EC”, which defines the requirements for a digital signature to be legal. See Wikipedia (<http://en.wikipedia.org/wiki/XAdES>) for a short description, but more importantly for the references to the standards (XMLDSig and XAdES itself).

This document assumes good knowledge about xml and basic knowledge about XMLDSig. The following sub-sections describe the requirements for XMLDSig and XAdES-BES and XAdES-T.

##### Structure

Xades is based on XMLDSig and comes in several variant. Of the 2 base variants, MyCareNet uses the Xades-BES. Of the different Xades extensions, only Xades-T is used. MyCareNet optionally uses the Manifest-element as defined by the XMLDSig standard.

This paragraph describes the different structures (very short) and indicates the specific requirements of MyCareNet.

###### XMLDSig

This section assumes you know that an xml digital signature is a “Signature” element that contains signed info, a signature value, key info and optionally other objects. It also assumes you understand the principal of references and transforms.

For xml digital signature the version of September 2000 with namespace “http://www.w3.org/2000/09/xmldsig#” must be used.

A XML digital signature has a canonicalization method and a signature method. For the canonicalization, exclusive without comments (http://www.w3.org/2001/10/xml-exc-c14n#) is advised. For signature RSASSA-PKCS1-v1\_5 with SHA-256 (http://www.w3.org/2001/04/xmldsig-more#rsa-sha256) is advised. RSASSA-PSS isn’t supported, but stronger hash methods like SHA-512 are supported.

The signature must contain a reference to the blob in it signed info. It is advised to have an id based reference (e.g. “#blob”. The id value can either be a fixed value (proposed value “blob”) or a UUID/GUID (e.g. “idF81E5355-7D18-4B94-A545-7B85A6A8FBFE”). The actual value is of no importance.

The reference must contain one or several of following transforms, in the below order:

* “http://www.w3.org/2000/09/xmldsig#base64”: to base64 decode the content of the message (always needed).
* “urn:nippin:xml:sig:transform:optional-deflate” to deflate the content of the message (needed if the content is deflated, i.e. for Async flows)
* “http://www.w3.org/2001/10/xml-exc-c14n#” in case the content is XML and no other formats like text or pdf.

The signature is only complete when it contains both the signature value and the key info. For the key info, only X509 certificates are allowed. It is required to add the entire chain and not just the signing certificate.

###### XAdES-BES

XAdES-BES is there to give legal meaning to the signature and provide a location to add technical objects to enforce it.

The targeted version of XAdES is 1.4.2, although for our use it is 100% compatible with version1.3.2. The namespace is [http://uri.etsi.org/01903/v1.3.2#](http://uri.etsi.org/01903/v1.3.2).

XAdES starts with an “object”-node in the signature. It contains a single “QualifyingProperties” that belongs to the XAdES namespace and references the signature via the “Target” attribute. For this the signature-node must have an “Id” attribute.

For XAdES-BES, only the “SignedProperties”-element is required. The “UnsignedProperties”-element is only needed for the XAdES-T (and other) XAdES variants. As there name indicates, only the signed properties are signed in the signature. Because of this, the unsigned properties can be changed afterward. The signed properties must have an Id for which there is a reference in the signature. This reference must have 1 transform: xml inclusive (http://www.w3.org/TR/2001/REC-xml-c14n-20010315).

The signed signature properties inside the signed properties element define the following elements:

* **Signing time**: the system time when the message was signed. This is purely informational, but useful, therefore it should be present.
* **Signing Certificate**: Must be present and have the information about the signing certificate as defined by the XAdES standard
* Since it is XAdES-BES and not XAdES-EPES, there should not be a **signature policy identifier**. Due to the specific content, it isn’t implicit and the explicit version isn’t formalized yet. In the future there might be one defined which should be added. In the meantime this value is ignored.
* **Signature production place:** This identifies the place where the document is signed and may be present.
* **Signing role:** this defines the role of the signer (e.g. hospital administrator). This should not be provided because the format to provide this isn’t defined yet.

None of the signed data object properties should be provided, but some are allowed

* **Data Object Format:** may be present and defines the format of the signed content
  + **Description:** may be present, but only informational
  + **Object Identifier:** should be present (if the parent is) and contains the URL to the message definition (e.g. <http://www.carenet.be/nl/activities/hopital/hospitalisation/messages_et_flux/>)
  + **Mime Type:** may be present must be the same as the content type of the “content type” attribute of the blob element (e.g. “text/plain”)
  + **Encoding:** should not be provided
  + **Object reference (attribute):** must be present (if the parent is) and indicates the reference that points to the blob.
* **Commitment type indicator:** may be present and defines what the party commits too.
  + **Commitment type id:** must be present (if the parent is) and must always be proof of origin ([http://uri.etsi.org/01903/v1.2.2#ProofOfOrigin](http://uri.etsi.org/01903/v1.2.2" \l "ProofOfOrigin)). The other values aren’t possible because they are part of the protocol.
* **Timestamps:** XAdES supports timestamps on the message itself, but this should not be provided. The only timestamp that should be provided is the one of the XAdES-T (see next section) because then it has a lot more meaning to when the message was created.

XAdES-BES also allows counter signatures in the unsigned signature properties, but this isn’t supported by NIP-PIN.

###### XAdES-T

XAdES-T extends XAdES-BES with a timestamp of the signature. That way it is known without any doubt when the message was created and when the signer committed to it. Keep in mind that the commitment of the signer is always “create, approved and sent” and not just “created”. Committing to have sent does not mean it is actually send, it only means the singer can’t claim it wasn’t.

According to the XAdES standard the timestamp must be calculated on the “SignatureValue” element. NIP-PIN requires that this timestamp must be generated by eHealth. Because the signature value is an xml-element, XAdES status that there must be a canonicalization executed on it, which should be exclusive without comments (http://www.w3.org/2001/10/xml-exc-c14n#). Because the eHealth timestamp authority does not support this, the call to the eHealth service should be with the hash-value so the canonicalization is done by the client. The hash method should be sha-256 or higher.

Since eHealth returns RFC3161 compliant timestamps, it must be an “encapsulated time stamp” in the XAdES signature timestamp.

Warning, currently eHealth returns a double base64 encoded timestamp which isn’t a valid value for XAdES-T. It is important to do an additional base 64 decoding of the returned timestamp before providing it to the XAdES engine.

###### Manifest

The manifest is an element with the same name in the “object” node and contains additional references. While the signed info of the XMLDSig contains a reference to the blob that was generated by the party, the manifest contains a reference to the blob generated by the other party.

This means it is only applicable on responses generated by the IO. The signed info of the XMLDSig contains a reference to the blob of the response, while the manifest contains a reference to the request of the care provider.

The requirements of the manifest reference are the same as the reference of the signed info. Therefore, the manifest reference in the response should be exactly the same (including the hash) as the signed info reference of the request.

The advantages of using a reference in a manifest instead of a reference directly in the signed info are the following:

* The request isn’t required for the validation of the response
* The request can be validated by standard libraries because manifest validation is part of the XMLDSig standard (no need to for custom implementations)
* Errors in the request do not break the probative force of the response (which is probably a reject of the request since it’s probative force was invalid)

##### Usage

This section describes how the XAdES is used and what parts are required.

###### Variants

The following combinations are used by NIP-PIN. See the service catalogue which variant is used (if used at all):

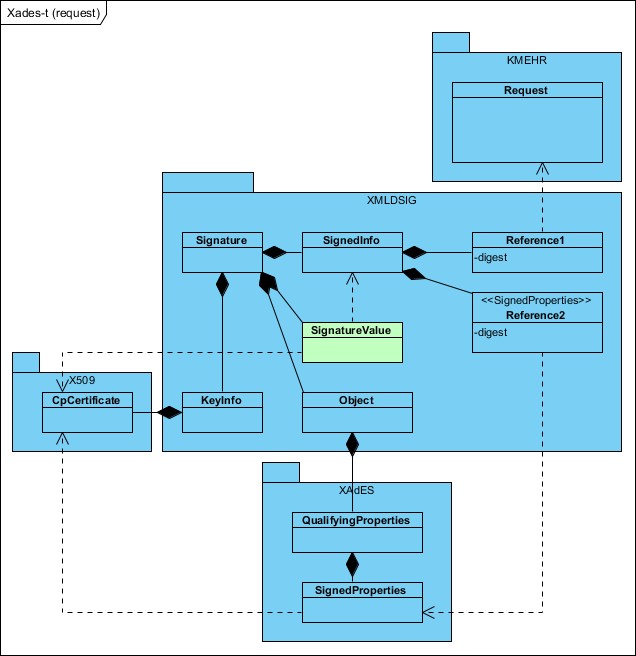
1. **XAdES-BES/XAdES-T with request ref**: The care provider must include the XAdES-BES (no timestamp) with its request, while the IO will return a XAdES-T with a reference to the request in its manifest. This way the timestamp of the response protects both response and request.
2. **XAdES-T/XAdES-T**: The care provide must include the XAdES-T with its request, while the IO will return a XAdES-T (without reference to the request).

Variant 1 only requires a single timestamp but still provides probative force of both the request and response. For the requestor it is important that he must keep the response in order to proof the probative force of its request (since he requires a timestamp).

Variant 2 as a little less complex (probative force of request only requires request) but requires 2 timestamps and therefore is less interesting.

###### Schemas

XAdES-BES variant becomes the following in an UML diagram:

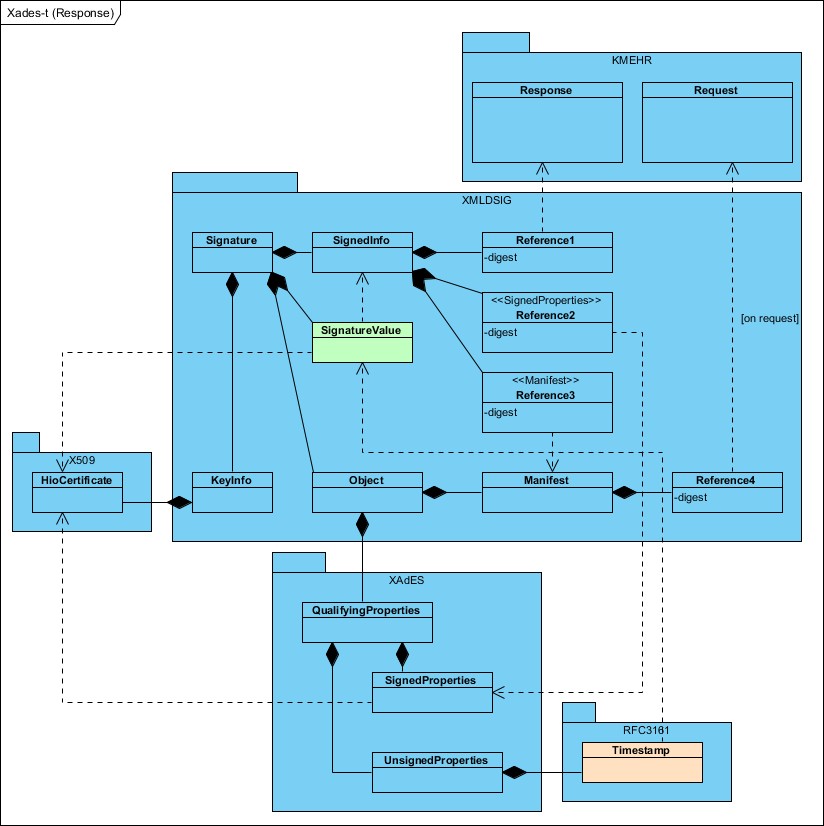


The signature is the root element and the signature value is calculated by the care provider with its own certificate. It is calculated on the signed info which contains:

* A reference (1) with the digest of the request after transforms. Therefore the signature protects the request.
* A reference (2) with the digest of the signed properties as defined by XAdES. These signed properties protect the care provider certificate; therefore the signature indirectly protects the care provider certificate.

There is not timestamp, so no probative force by its own.

XAdES-T variant with request reference becomes the following in an UML diagram:



The signature is the root element and the signature value is calculated by the HIO with its own certificate. It is calculated on the signed info which contains:

* A reference (1) with the digest of the response after transformation. Therefore the signature directly protects the response.
* A reference (2) with the digest of the signed properties as defined by XAdES. These signed properties protect the care provider certificate. Therefore the signature indirectly protects the HIO certificate.
* A reference (3) with the digest of the manifest. The manifest contains a reference (4) with the digest of the request after transformations. Therefore the signature protects the request indirectly via the manifest.

The HIO puts a timestamp on the digest of the signature value. This means that both request and response are protected by the same timestamp.

###### Producing a XAdES of a Request

When sending a request, a XAdES-BES (or XAdES-T) must be created. This section describes how to do this, assuming you have a XAdES library but without specifying which implementation.

Before you can create a XAdES you need an XML document, most likely as a DOM object. For this you create a new document, containing a (any) root element (e.g. “root” or “wrap”). The request must be added as a child of this root element. The local name, namespace and structure aren’t important, only the “Id” attribute must be present and have the same value as the “Id”-field of the “Detail”-field of the web service request. The request must also have content, which must be the same raw content as the “detail” of the web service request and in the same format[[1]](#footnote-1). Don’t forget that your web service client probably will do the base64, while this will most likely not be the case when you create your XML document, so you have to do it yourself when signing but omit it when sending.

On this (dummy) XML document you must calculated a XAdES compliant signature. You must add a reference that references the request-element via its Id and specify the required transformations (see 2.8.3.5.1.1). You probably need to configure the XAdES library to use the required transformation for the reference required by XAdES itself. If needed, the XAdES library must also be configured to understand the XAdES-T compliant timestamp that must be obtained from eHealth. This will probably require an extension of the library in order to support the TSA interface offered by eHealth.

If the deflate transform is used, the XAdES library will most likely not support this out of the box since it isn’t a standard transform. With a decent XAdES library, it should be easy to add this transform according to the specs: take the input as bytes and inflate it (using DEFLATE, not ZLIB, GZIP or PKZIP).

###### Validating a XAdES of a Response

When receiving a response with a XAdES-T, possibly with manifest reference to the request, it should be validated. For good measures, the manifest reference should also be validated but does not impact the validity of the response (it only indicates that request was invalid).

As with the request, an XML document must be create. It also has a (any) root element. It should contain the response as a child of the root element. As with the request, only “Id” and raw content are relevant. It is best to also add the request as a second child of the root element, this is only required if it is referenced and you want to validate it but there is not harm in doing it always. Last but not least, the XAdES-T of the HIO must be added as a child of the root element. You must take care not to alter the XAdES-T in such a way that it breaks the signatures, what this means depends on the library.

Once the XML document is created, you can validate the XAdES signature in the way the library foresees. There is no need for much configuration since most of the required information is part of the XAdES signature. You will have to register the deflate transform (take the input as byes and inflate it as standard DEFLATE, not ZLIB, GZIP or PKZIP). It is also advised to instruct the library to validate the manifest libraries.

The library will indicate if the signature was valid and will return information about signature, its signer and the manifest validation. A valid signature isn’t sufficient; you must also make sure the signature is: XAdES-T (and not just XAdES-BES), that time of the signature is acceptable (business dependent) and you must validate the signer certificates via PKI, including CRL or OCSP. You do not have to validate the signer is an HIO; this is implicitly enforced by the fact that the response comes from NIP-PIN. Some of the checks may be done by the library, others might require custom development.

The result of the validation may also contain the result of the manifest validation. The manifest validations do not impact the result the response validation. The manifest validation is only needed to validate the probative force of the request; it does require the response to be valid though.

##### Example

Below is an example of an xml as can be used to *validate* the response of the HIO and the request of the care provider. It is as correct as possible, but can’t be used to test it only provides an idea of what to expect.

<?xml version="1.0" encoding="utf-8"?>

<wrap xmlns="urn:dummy">

<response Id="ioxxx">c03OyFdIrUjMLchJBQA=</response>

<request Id="blob">c03OyFdIrUjMLchJBQA=</request>

<Signature Id="xades"

xmlns="http://www.w3.org/2000/09/xmldsig#">

<SignedInfo>

<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />

<SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />

<Reference Id="blobref" URI="#ioxxx">

<Transforms>

<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64" />

<Transform Algorithm="urn:nippin:xml:sig:transform:optional-deflate" />

</Transforms>

<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />

<DigestValue>A7ogTlDRJuRnTABeBNguhMITZngK8fQ71Uo3gWtqs0A=</DigestValue>

</Reference>

<Reference URI="#idSignedProperties"

Type="http://uri.etsi.org/01903#SignedProperties">

<Transforms>

<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />

</Transforms>

<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />

<DigestValue>wkIhggTCbOfWC5e7bMlbk3D+nD0W7hCwde2XIxwWzKs=</DigestValue>

</Reference>

<Reference URI="#manifest" Type="http://www.w3.org/2000/02/xmldsig#Manifest">

<Transforms>

<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>

</Transforms>

<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>

<DigestValue>345x3rvEPO0vKtMup4NbeVu8ZngK8fQ71Uo3gWtqs0A=</DigestValue>

</Reference>

</SignedInfo>

<SignatureValue>

<!-- snip base64 encoded signature value -->

</SignatureValue>

<KeyInfo>

<X509Data>

<X509Certificate>

<!-- snip: base64 encoded signing certificate (eHealth/eID) -->

</X509Certificate>

<X509Certificate>

<!-- snip: base64 encoded intermediate certificate (e.g. government CA) -->

</X509Certificate>

<X509Certificate>

<!-- snip: base64 encoded root certificate (belgium root ca) -->

</X509Certificate>

</X509Data>

</KeyInfo>

<Object>

<Manifest Id="manifest">

<Reference URI="#blob">

<Transforms>

<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>

</Transforms>

<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>

<DigestValue>A7ogTlDRJuRnTABeBNguhMITZngK8fQ71Uo3gWtqs0A=</DigestValue>

</Reference>

</Manifest>

<QualifyingProperties Target="#xades"

xmlns="http://uri.etsi.org/01903/v1.3.2#"

xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

<SignedProperties Id="idSignedProperties">

<SignedSignatureProperties>

<SigningTime>2011-04-26T14:54:46Z</SigningTime>

<SigningCertificate>

<Cert>

<CertDigest>

<ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />

<ds:DigestValue>SJ2WmoWlQHEulei0+1LMynZHc7Agg7h3kIi7tmCH+Yw=</ds:DigestValue>

</CertDigest>

<IssuerSerial>

<ds:X509IssuerName>SERIALNUMBER=201001, CN=Citizen CA, C=BE</ds:X509IssuerName>

<ds:X509SerialNumber>21267647932559755252544023651377138403</ds:X509SerialNumber>

</IssuerSerial>

</Cert>

</SigningCertificate>

</SignedSignatureProperties>

</SignedProperties>

<UnsignedProperties>

<UnsignedSignatureProperties>

<SignatureTimeStamp>

<ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />

<EncapsulatedTimeStamp>

<!-- snip: base64 encoded eHealth timestamp -->

</EncapsulatedTimeStamp>

</SignatureTimeStamp>

</UnsignedSignatureProperties>

</UnsignedProperties>

</QualifyingProperties>

</Object>

</Signature>

</wrap>

### Message variants

As indicated before, NIP-PIN never requests the same information in 2 different elements. In many common cases 2 elements that can have a different value do have actually have the same value.

Below are the 2 different ways this can be represented in the request to NIP-PIN. The package provider can select any of these variants.

#### Basic

For the basic variant you simply need to specify the values multiple times. This is the only variant that is compatible with the DataContract serializer of WCF and therefore it is the variant represented by the online WSDL of the services.

When you create a request via the generated classes of the WSDL, this is probably the easiest way. You simply fetch the value once and assign it to the different properties. Here is an example of both the resulting XML and the code (C#) to create it.

|  |  |
| --- | --- |
| OrigineType o = new OrigineType();  o.CareProvider = new CareProviderType();  o.CareProvider.Organisation = new IdType();  o.CareProvider.Organisation.Name = "TestLabo1";  o.CareProvider.Organisation.Nihii =  new NihiiType();  o.CareProvider.Organisation.Nihii.Quality =  "labo";  o.CareProvider.Organisation.Nihii.Value =  "89999568803";  o.CareProvider.Nihii =  o.Sender.Organisation.Nihii; | <Origin>  <CareProvider>  <Nihii>  <Quality>labo</Quality>  <Value>89999568803</Value>  </Nihii>  <Organisation>  <Name>TestLabo1</Name>  <Nihii>  <Quality>labo</Quality>  <Value>89999568803</Value>  </Nihii>  </Organisation>  </CareProvider>  </Origin> |

#### Referenced

There is an alternative WSDL that is annexed to this document that allow one element to refer to another element of value of an element. This isn’t compatible with the Data Contract Serializer of WCF and although it is possible with JAX-WS and XML Serializer of WCF it is probably easier to use the basic variant.

This variant is intended for more complex applications that do generate (parts) of the request directly in xml and use a non-business specific web service client for NIP-PIN. In many cases a package can even have one (or several) common inputs that can be included in every request without any modifications.  
Both WCF and JAX-WS generate generic web service client (but by default generate business specific clients). This message variant is intended for this generic web service client. Important to know is that NIP-PIN does not provide any support on the usage of the generic web service client, only on the output of the common input xml.

The references are simple IDs in the same document (i.e. no XPath expressions).

1. The raw content is the xml or flat as required by the service, only this must be exactly the same between the XML and the web service. On this the same formatting must be applied, but these formattings can have different config. For example, you can deflate with “best speed” for XAdES but use “best compression” for the web service. [↑](#footnote-ref-1)