XEP-0518: Payment Required

JC Brand
mailto:jc@opkode.com
xmpp:jc@opkode.com

2026-08-25
Version 0.1.0

Status	Type	Short Name
Experimental	Standards Track	NOT_YET_ASSIGNED

This specification defines an XMPP protocol extension that enables services to require payment before granting access to a resource. It provides a payment-system-neutral invoice format built on payment URIs (such as the RFC 8905 payto scheme), supporting multiple concurrent payment options, from bank transfers to transfers over distributed ledgers or instant-settlement networks, and integrates with the existing CAPTCHA challenge mechanism defined in XEP-0158.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction **1**

2 Requirements **2**

3 Glossary **3**

4 Use Cases **3**
- 4.1 Paid MUC Room Entry with Multiple Payment Options . . . 3
- 4.2 Per-Message Fee for an AI Agent Bot . . . 5
- 4.3 Anti-Spam Deposit for Non-Roster Contacts . . . 7
- 4.4 Proactive Invoice Request . . . 7
- 4.5 Payment Challenge as Alternative to CAPTCHA . . . 8

5 Protocol **10**
- 5.1 Namespace . . . 10
- 5.2 Elements . . . 10
  - 5.2.1 The invoice Element . . . 10
  - 5.2.2 The option Element . . . 10
  - 5.2.3 The payment Element . . . 11
  - 5.2.4 The payment-required Element . . . 12
  - 5.2.5 The receipt Element . . . 13
  - 5.2.6 The get-invoice Element . . . 14
- 5.3 Error Flow . . . 14
- 5.4 Verification . . . 15

6 Business Rules **15**
- 6.1 Service Behavior . . . 15
- 6.2 Client Behavior . . . 16

7 Implementation Notes **16**

8 Accessibility Considerations **17**

9 Internationalization Considerations **17**

10 Security Considerations **18**
- 10.1 Replay Attacks . . . 18
- 10.2 Session Binding . . . 18
- 10.3 Invoice Tampering . . . 18
- 10.4 Amount Presentation Integrity . . . 19
- 10.5 Financial Safety . . . 20

11 Privacy Considerations **20**

12 IANA Considerations **20**

13 XMPP Registrar Considerations **20**
13.1 Protocol Namespaces 20
13.2 Service Discovery Features 21
13.3 Payment Schemes and Proof Types 21
13.4 Invoice Request Service Registry 21

14 Design Considerations **22**
14.1 Using Data Forms for the Invoice 22
14.2 Choosing auth as the Error Type 22
14.3 Streaming Payment Sessions 23

15 XML Schema **23**

1 Introduction

This XEP provides a mechanism for XMPP services to require payment before access to their resources or functionality is granted. Use cases include charging for account registration or hosting, compensating operators of AI agents or bots for compute costs, monetizing file hosting services per upload or download, requiring an entry fee for access to a Multi-User Chat (XEP-0045) [1] room, and requiring a small payment from users not already in a contact's roster as a proof-of-intent mechanism to deter unsolicited messages.
Existing XMPP anti-spam work (CAPTCHA Forms (XEP-0158) [2]) defines robot challenges using either image/audio CAPTCHAs or hashcash proof-of-work tokens. This specification extends that concept by adding a payment-required challenge type that follows the same challenge-retry flow, while independently defining a richer invoice format suitable for both human and automated payers.
Protocols such as Publish-Subscribe (XEP-0060) [3] already acknowledge that services may require payment for operations (e.g., node subscriptions or item retrieval) but leave the payment mechanism out-of-band. This specification provides the in-band protocol to fulfill that need.
The design is informed by analogous protocols in the HTTP ecosystem: the long-reserved HTTP 402 Payment Required status code; the L402 protocol published by Lightning Labs, in which a client receives a WWW-Authenticate header containing a macaroon token and a Lightning invoice and presents a token-preimage credential after payment; the x402 protocol published by Coinbase et al., in which a client receives an HTTP 402 response with a PAYMENT-REQUIRED header listing accepted payment schemes and retries with a PAYMENT-SIGNATURE header; and the Machine Payments Protocol (MPP) published by Stripe and Tempo, an IETF-tracked open standard for machine-to-machine payments that similarly uses HTTP 402 challenges with multi-method selection, machine-readable error codes, payment receipts, and cryptographic challenge binding.
The conceptual parallels shared by all four are:

1. a service declines a request and presents payment instructions in-band
2. the client completes payment and retries, optionally supplying a proof-of-payment token
3. multiple simultaneously valid payment options MAY be presented so that the client can choose the most suitable scheme

This specification is payment-system agnostic. A service MAY require payment exclusively via bank transfer, distributed ledger, instant-settlement network, or any combination thereof. No payment system is privileged over any other at the protocol level. This specification also does not enumerate or curate payment systems. Each payment option is expressed as a

[1] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[2] XEP-0158: CAPTCHA Forms <https://xmpp.org/extensions/xep-0158.html>.
[3] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.

payment URI, and the payment system is identified by that URI's scheme. Adding support for a new payment network therefore requires no change to this document.
The payment URI SHOULD be registered in the IANA URI Schemes registry, but there are URIs widely used by wallet and payment software that are not yet registered, such the lightning: URI scheme, documented in the Lightning Network specifications (BOLT #11). For the payto: scheme, which is IANA registered, the payment target types are maintained externally per RFC 8905.

2 Requirements

This specification was designed with the following requirements in mind:

- A service MUST be able to decline a stanza and return structured payment instructions using a standard XMPP error stanza.
- The payment instructions MUST support multiple concurrent payment options, each carrying sufficient information for fully automated processing as well as human-readable presentation.
- Each payment option MUST be expressed as a payment URI, identified by its URI scheme, so that the specification need not maintain its own registry of payment systems.
- A client MUST be able to support this protocol without parsing scheme-specific payment URIs and therefore MAY treat a payment URI as an opaque string to be handed over to a payment application (or rendered as a QR code).
- A payment option MAY additionally carry a pre-formatted QR payload for payment systems whose scannable QR format differs from the URI. Clients SHOULD render it as a QR code without interpreting it.
- Proof-of-payment tokens MUST be expressible in-band so that a client can retry a declined stanza with evidence of payment.
- The protocol MUST integrate gracefully with the existing CAPTCHA Forms (XEP-0158) [4] challenge mechanism.
- The protocol MUST be usable at the stanza level independently of the application layer (MUC, file upload, message routing, etc.) so that application-layer XEPs can reference this specification without modification.
- Services MUST be able to advertise support via Service Discovery (XEP-0030) [5].

[4] XEP-0158: CAPTCHA Forms <https://xmpp.org/extensions/xep-0158.html>.
[5] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

3 Glossary

Payer The XMPP entity that receives a payment-required error and is expected to fulfill the payment. Typically this is the originating client.

Payee / Service The XMPP entity that requires payment before granting access to a resource. This may be a server component, a MUC service, a bot, or any other XMPP service.

Invoice A structured XML element, defined in this specification, that conveys one or more payment options to a payer.

Payment Option A single, self-contained description of one method by which payment may be made (e.g., a SEPA bank transfer, a UPI payment request, a Lightning Network invoice).

Proof of Payment A token supplied by the payer on retry to demonstrate that a payment has been made. The format depends on the payment scheme used (e.g., a Lightning Network payment preimage, a bank transaction reference number).

Payment Session A short-lived, service-generated identifier that correlates a payment-required error with the subsequent retry stanza, analogous to a session identifier in Ad-Hoc Commands (XEP-0050) XEP-0050: Ad-Hoc Commands <https://xmpp.org/extensions/xep-0050.html>..

4 Use Cases

4.1 Paid MUC Room Entry with Multiple Payment Options

Juliet operates a premium conference room on conference.shakespeare.lit that charges an entry fee. Romeo attempts to join but is declined and presented with multiple invoice options.

Listing 1: Romeo attempts to join a paid MUC room

```
<presence from='romeo@shakespeare.lit/orchard'
          to='upperroom@conference.shakespeare.lit/Romeo'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>
```

Listing 2: The MUC service declines with a payment-required error

```
<presence from='upperroom@conference.shakespeare.lit'
          to='romeo@shakespeare.lit/orchard'
          type='error'>
  <error type='auth'>
    <payment-required xmlns='urn:xmpp:payment:0'/>
    <text xml:lang='en'
          xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>
      An entry fee of EUR 5.00 is required to join this room.
```

```
    </text>
    <invoice expires='2026-04-19T14:00:00Z'
             purpose='Premium room monthly entry'
             session='c4a1f902-7d3b-4e8c-a510-2f9b0e6d3178'
             xmlns='urn:xmpp:payment:0'>
      <description>
        Pay once per calendar month. Include your JID as the payment
        reference so your access can be confirmed promptly.
      </description>
      <option label='Bank transfer (SEPA)'
              proof='reference'>payto://iban/DE02200400300200270112?
                  amount=EUR:5.00&amp;message=upperroom-c4a1f902&amp;
                  receiver-name=Capulet+Hosting
        <qr>BCD
002
1
SCT

Capulet Hosting
DE02200400300200270112
EUR5.00

upperroom-c4a1f902</qr>
      </option>
      <option label='UPI'
              proof='reference'>payto://upi/capulethosting@examplebank
                  ?amount=INR:450&amp;message=upperroom-c4a1f902&amp;
                  receiver-name=Capulet+Hosting
      </option>
      <option label='Lightning Network (instant)'
              proof='preimage'>
                  lightning:lnbc50n1pn2s4czpp5qqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqypq

      </option>
    </invoice>
  </error>
</presence>
```

Listing 3: Romeo retries after paying by SEPA bank transfer

```
<presence from='romeo@shakespeare.lit/orchard'
          to='upperroom@conference.shakespeare.lit/Romeo'>
  <x xmlns='http://jabber.org/protocol/muc'/>
  <payment session='c4a1f902-7d3b-4e8c-a510-2f9b0e6d3178'
           xmlns='urn:xmpp:payment:0'>
    <proof type='reference'>NOTPROVIDED20260419DE02</proof>
  </payment>
</presence>
```

Because SEPA bank transfers may take time to settle, the service SHOULD set a generous session expiry for bank-transfer options and MAY grant access provisionally upon receiving the retry stanza, subsequently revoking it if reconciliation fails. Alternatively, where the service has detected the payment but has not yet confirmed settlement, it MAY ask the payer to wait and retry later (see Error Flow).

Listing 4: The service has seen the transfer but awaits settlement, asking Romeo to retry later

```
<presence from='upperroom@conference.shakespeare.lit'
          to='romeo@shakespeare.lit/orchard'
          type='error'>
  <error type='wait'>
    <payment-required xmlns='urn:xmpp:payment:0'
                      reason='payment-pending'
                      retry-after='3600'/>
    <text xml:lang='en'
          xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>
      Your payment has been received and is awaiting settlement.
      Please retry in about an hour.
    </text>
  </error>
</presence>
```

4.2 Per-Message Fee for an AI Agent Bot

Balthasar operates a news-summary bot that charges per query. Because the amounts are small and latency matters, only instant-settlement options are offered. This is a service-level policy decision, not a protocol constraint.

Listing 5: Romeo sends a message to the bot

```
<message from='romeo@shakespeare.lit/orchard'
         id='msg-001'
         to='newssummary@bots.shakespeare.lit'
         type='chat'>
  <body>Summarize today's news from Verona.</body>
</message>
```

Listing 6: The bot declines with a payment-required error

```
<message from='newssummary@bots.shakespeare.lit'
         id='msg-001'
         to='romeo@shakespeare.lit/orchard'
         type='error'>
  <body>Summarize today's news from Verona.</body>
  <error type='auth'>
    <payment-required xmlns='urn:xmpp:payment:0'/>
```

```
␣␣␣␣<text␣xml:lang='en'
␣␣␣␣␣␣xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>
␣␣␣␣␣␣A␣payment␣of␣10␣sats␣is␣required␣per␣query.
␣␣␣␣</text>
␣␣␣␣<invoice␣expires='2026-03-19T14:20:00Z'
␣␣␣␣␣␣␣␣␣␣␣␣purpose='Per-query fee'
␣␣␣␣␣␣␣␣␣␣␣␣session='a3f7c291-84d0-4b2e-9b1a-0f3e2d1c5678'
␣␣␣␣␣␣␣␣␣␣␣␣xmlns='urn:xmpp:payment:0'>
␣␣␣␣␣␣<option␣label='Lightning (instant, automated)'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣proof='preimage'>
    lightning:lnbc100n1pn2s3dzpp5qqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqypq
␣␣␣␣␣␣</option>
␣␣␣␣␣␣<option␣label='BOLT 12 offer'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣proof='preimage'>
    lightning:lno1qgsyz4uzesxqcyq5rqwzqfqpqdynf
␣␣␣␣␣␣</option>
␣␣␣␣</invoice>
␣␣</error>
</message>
```

Listing 7: Romeo retries after paying via Lightning

```
<message from='romeo@shakespeare.lit/orchard'
         id='msg-002'
         to='newssummary@bots.shakespeare.lit'
         type='chat'>
  <body>Summarize today's␣news␣from␣Verona.</body>
␣␣<payment␣session='a3f7c291-84d0-4b2e-9b1a-0f3e2d1c5678'
␣␣␣␣␣␣␣␣␣␣␣xmlns='urn:xmpp:payment:0'>
␣␣␣␣<proof␣type='preimage'>
    a8f3e1d2b4c9078564fae012cc3d99a1b5e7d0f3a2c81496057832bd7e4f0c1a</
    proof>
␣␣</payment>
</message>
```

Listing 8: The bot fulfills the request and returns a payment receipt

```
<message from='newssummary@bots.shakespeare.lit'
         id='msg-003'
         to='romeo@shakespeare.lit/orchard'
         type='chat'>
  <body>Today in Verona: the Montagues and Capulets have agreed to a
     temporary truce...</body>
  <receipt reference='
     a8f3e1d2b4c9078564fae012cc3d99a1b5e7d0f3a2c81496057832bd7e4f0c1a
     '
           session='a3f7c291-84d0-4b2e-9b1a-0f3e2d1c5678'
           settled='2026-03-20T09:01:14Z'
           xmlns='urn:xmpp:payment:0'/>
</message>
```

4.3 Anti-Spam Deposit for Non-Roster Contacts

Juliet's server requires a small deposit from senders not present in her roster, as a proof-of-intent mechanism to deter unsolicited messages. Instant-settlement options are used because slow bank transfers would allow queuing before verification.

Listing 9: An unknown sender is declined pending a deposit

```
<message from='juliet@shakespeare.lit'
         id='spam-001'
         to='stranger@othello.lit/mobile'
         type='error'>
  <body>Buy cheap goblets!</body>
  <error type='auth'>
    <payment-required xmlns='urn:xmpp:payment:0'/>
    <text xml:lang='en'
          xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>
      A small deposit is required to message this user for the first
          time.
      The deposit will be refunded if the recipient accepts your
          contact request.
    </text>
    <invoice expires='2026-03-19T16:01:00Z'
             purpose='First-contact␣anti-spam␣deposit'
             session='f1c3d9e2-7a04-4b8f-a629-3e0d15b7c412'
             xmlns='urn:xmpp:payment:0'>
      <option label='Lightning␣deposit'
              proof='preimage'>
                  lightning:lnbc100n1pn2s5azpp5qqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqypq

      </option>
    </invoice>
  </error>
</message>
```

4.4 Proactive Invoice Request

A client MAY proactively request an invoice from a service before sending the gated stanza. This is useful when the client wishes to present payment options to the user before committing to an action. Before sending a proactive invoice request, the client SHOULD discover whether the service supports this feature by querying for the 'urn:xmpp:payment:0#invoice-request' feature via Service Discovery (XEP-0030) [6].

Listing 10: Romeo requests an invoice before attempting to join

[6] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

```
<iq from='romeo@shakespeare.lit/orchard'
    id='inv-001'
    to='conference.shakespeare.lit'
    type='get'>
  <get-invoice service='muc-entry'
               target='upperroom@conference.shakespeare.lit'
               xmlns='urn:xmpp:payment:0'/>
</iq>
```

Listing 11: The service returns an invoice

```
<iq from='conference.shakespeare.lit'
    id='inv-001'
    to='romeo@shakespeare.lit/orchard'
    type='result'>
  <invoice expires='2026-04-19T14:00:00Z'
           purpose='Premium room monthly entry'
           session='3c91b2e4-6f07-4a2d-b839-5e0f17d9c823'
           xmlns='urn:xmpp:payment:0'>
    <option label='Bank transfer (SEPA)'
            proof='reference'>payto://iban/DE02200400300200270112?
               amount=EUR:5.00&amp;message=3c91b2e4&amp;receiver-name
               =Capulet+Hosting
    </option>
    <option label='Lightning (instant)'
            proof='preimage'>
               lightning:lnbc50n1pn2s4czpp5qqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqypq

    </option>
  </invoice>
</iq>
```

If the service does not support proactive invoice requests, it SHOULD return a <feature-not-implemented/> error of type "cancel".

4.5 Payment Challenge as Alternative to CAPTCHA

This specification extends CAPTCHA Forms (XEP-0158) [7] by defining a new challenge field type that carries a payment invoice, allowing a service to offer payment as an alternative to solving a CAPTCHA.
To embed a payment invoice within a CAPTCHA Forms (XEP-0158) [8] challenge, the challenger SHOULD include a field with var='urn:xmpp:payment:0' inside the Data Form. The invoice is transmitted as a sibling element to the <x/> element in the same stanza.

[7] XEP-0158: CAPTCHA Forms <https://xmpp.org/extensions/xep-0158.html>.
[8] XEP-0158: CAPTCHA Forms <https://xmpp.org/extensions/xep-0158.html>.

Listing 12: A service offers payment as an alternative to a CAPTCHA

```
<message from='security.shakespeare.lit'
         id='challenge-001'
         to='romeo@shakespeare.lit/orchard'>
  <x type='form' xmlns='jabber:x:data'>
    <title>Verify your intent</title>
    <field type='hidden' var='FORM_TYPE'>
      <value>urn:xmpp:captcha</value>
    </field>
    <field type='hidden' var='challenge'>
      <value>challenge-abc-123</value>
    </field>
    <field label='Enter␣the␣text␣you␣see:'
           type='text-single'
           var='ocr'>
      <media xmlns='urn:xmpp:media-element'>
        <uri type='image/jpeg'>https://security.shakespeare.lit/
            captcha/abc123.jpg</uri>
      </media>
      <required/>
    </field>
    <field label='Or␣pay␣the␣verification␣fee␣(enter␣session␣ID␣after␣
       payment):'
           type='text-single'
           var='urn:xmpp:payment:0'/>
  </x>
  <invoice expires='2026-03-19T14:30:00Z'
           purpose='Account␣verification␣fee'
           session='9d4e2c01-5b8f-4a3e-b796-0f1e28d7c589'
           xmlns='urn:xmpp:payment:0'>
    <option label='Bank␣transfer␣(SEPA)'
            proof='reference'>payto://iban/DE02200400300200270112?
               amount=EUR:0.01&amp;message=9d4e2c01
    </option>
    <option label='Lightning'
            proof='preimage'>
               lightning:lnbc10p1pn2s6azpp5qqqsyqcyq5rqwzqfqqqsyqcyq5rqwzqfqypq

    </option>
  </invoice>
</message>
```

5 Protocol

5.1 Namespace

This specification uses the namespace 'urn:xmpp:payment:0'. (see Namespace Versioning regarding the possibility of incrementing the version number)

5.2 Elements

The following elements are defined in the 'urn:xmpp:payment:0' namespace.

5.2.1 The invoice Element

The <invoice/> element is the container for all payment options associated with a single payment event. It MUST contain at least one <option/> child element and MAY contain a <description/> child element whose character data provides additional human-readable context.
The <invoice/> element has the following attributes:

- session (REQUIRED) — An opaque, service-generated string that MUST be globally unique and MUST be included verbatim in the 'session' attribute of the corresponding <payment/> element. How the service generates this value is an implementation choice; Session Binding describes two strategies (a random identifier such as a UUID version 4 per RFC 4122 with server-side state, or a stateless keyed token derived from the invoice parameters). Where a bank transfer option is present, the session value also serves as the RECOMMENDED payment reference that the payer SHOULD include in the transfer.
- expires (RECOMMENDED) — A UTC date-time value, formatted per XMPP Date and Time Profiles (XEP-0082) [9], after which the invoice MUST NOT be honored. The payer SHOULD NOT attempt payment after this time.
- purpose (OPTIONAL) — A short human-readable string describing the reason for the payment requirement (e.g., "Room entry fee", "File upload", "Anti-spam deposit").

5.2.2 The option Element

Each <option/> element describes one complete, self-contained payment method. The payer MUST use exactly one option to fulfill the invoice.
The character data of the <option/> element MUST be a **payment URI**. The payment system is identified by the URI's scheme (e.g. payto: per RFC 8905), and the amount, currency, and beneficiary are carried within the URI as defined by that scheme's own standard. This specification neither defines nor curates the set of usable schemes. The payto URI scheme

[9] XEP-0082: XMPP Date and Time Profiles <https://xmpp.org/extensions/xep-0082.html>.

(RFC 8905) is RECOMMENDED as the primary scheme.
A client is NOT required to parse the payment URI. It MAY treat the URI as an opaque string to be handed to an external payment application or rendered as a QR code; see Amount Presentation Integrity.
The <option/> element possesses the following attributes:

- proof (OPTIONAL) — The type of proof-of-payment the service expects the payer to echo on retry (see The payment Element). The issuer declares this value and the payer returns it verbatim in the 'type' attribute of the <proof/> element. Its absence means the service verifies payment out of band and requires no payer-supplied proof. The value's meaning is defined by the payment system the option uses; Conventional Proof Types lists common values non-normatively.
- label (OPTIONAL) — A short human-readable string for display in a list (e.g., "Bank transfer (SEPA)", "UPI", "Lightning").

This specification does not define a separate human-readable amount field. The amount is carried in the payment URI. A client that parses the option's scheme reads it from there for display, while one that does not delegates the URI to a payment application that displays it (see Amount Presentation Integrity). A duplicate amount supplied alongside the URI would be redundant in the first case and unverifiable in the second. The amount is readily extracted from common schemes (payto from its amount=currency:value parameter, bitcoin: from its decimal amount, a lightning: BOLT 11 invoice from its prefix). For less common schemes, and where no amount is encoded at all (such as a zero-amount Lightning invoice), the payer's payment application presents the amount it is about to send for confirmation.
The <option/> element MAY also contain a <qr/> child element whose character data is a payment string to be rendered as a QR code. This is useful where the payment URI is not itself the QR payload that the payer's payment application expects to scan, for example, a SEPA credit transfer for which banking apps scan a European Payments Council (EPC069-12) payload rather than a payto URI. A client renders the <qr/> content as a QR code *without interpreting it*, exactly as it would QR-encode the URI, so no scheme-specific parsing is required. This specification does not define or constrain the format of the <qr/> payload; that is a matter between the service and the scanning payment application. Because a payment URI is always present, the <qr/> element is purely an additional rendering and payment convenience and is subject to the same integrity considerations as other invoice content (see Amount Presentation Integrity). Clients SHOULD also present the underlying URI as selectable or copyable text for accessibility.

5.2.3 The payment Element

The <payment/> element is added by the payer as an extension to the retried stanza to indicate that a payment has been made for the referenced session.
The <payment/> element possesses a single attribute:

- session (REQUIRED) — The value of the 'session' attribute from the <invoice/> element that this payment satisfies. The session is the sole correlator the service needs. Because the service issued the invoice for that session, it already knows which option(s) and payment system(s) were offered.

The <payment/> element MAY contain a <proof/> child element. The <proof/> element possesses a required 'type' attribute and its character data is the proof token. Where the satisfied option carried a 'proof' attribute, the payer SHOULD set 'type' to that declared value. The proof token is opaque to XMPP; its syntax and verification are defined by the payment system the option uses, not by this specification.
The following proof types are in common use. This list is **non-normative**: it is a convenience vocabulary, not a registry maintained by this specification. The format and verification of each is defined by the relevant payment system's own standard.

Type	Meaning
preimage	A secret revealed upon settlement of a hash-locked payment (e.g. a Lightning Network payment preimage).
reference	A transaction or end-to-end reference string supplied by the payment network (e.g. a SEPA end-to-end identifier, a UPI transaction ID, a PIX E2EID).
txid	An on-chain transaction identifier.
proof	A payee-verifiable cryptographic payment proof supplied out of band (e.g. a private-network transaction proof).
token	An opaque bearer token issued by a facilitator after verified payment, enabling a "pay once, use many times" pattern.

5.2.4 The payment-required Element

The <payment-required/> element is an application-specific stanza error condition. It is used as a child of the <error/> element to indicate that the requested action requires payment before it can be fulfilled.
The <payment-required/> element possesses the following OPTIONAL attributes:

- reason (OPTIONAL) — A machine-readable code providing additional detail. This is intended for use in error responses to retried stanzas (i.e., where a <payment/> was present), not in the initial decline. Defined values are:

Value	Meaning
payment-required	No <payment/> element was present. The resource requires payment before access is granted.
invalid-session	The 'session' value is unknown, has already been consumed, or does not match the HMAC verification check.

Value	Meaning
payment-expired	The invoice has passed its 'expires' time and may no longer be honored. The payer SHOULD request a fresh invoice.
verification-failed	A <proof/> element was present but the proof could not be verified as valid for the referenced payment.
payment-insufficient	The payment amount detected was less than the required amount.
payment-pending	A payment has been detected for the referenced session but has not yet been confirmed as settled. The payer should wait and retry. This is not a failure and is therefore paired with stanza error type "wait" and SHOULD be accompanied by a 'retry-after' attribute (see Error Flow).

The <payment-required/> element additionally possesses an OPTIONAL retry-after attribute. If it exists, it MUST be set to a non-negative integer number of seconds the payer SHOULD wait before retrying the same session. It is meaningful with the 'payment-pending' reason (and MAY accompany any transient condition returned with stanza error type "wait").

5.2.5 The receipt Element

The <receipt/> element MAY be included by a service in the successful response to a retried stanza to provide the payer with a machine-readable record of the settled payment. It is an OPTIONAL protocol element and its absence does not indicate that the payment failed.
The <receipt/> element has the following attributes:

- session (REQUIRED) — The session identifier from the <invoice/> that was satisfied, allowing the payer to correlate the receipt with the original invoice.
- reference (RECOMMENDED) — A payment-network reference that can be used for auditing and dispute resolution. For Lightning Network payments this is, for example, the payment hash; for bank transfers, the end-to-end identifier provided by the receiving bank; for distributed ledger payments, the transaction identifier.
- settled (OPTIONAL) — A UTC date-time value, formatted per XMPP Date and Time Profiles (XEP-0082) [10], indicating when the payment was confirmed as settled. For asynchronous settlement methods (e.g., bank transfers) this MAY be omitted if settlement has not yet been confirmed.

A service that supports emitting receipts SHOULD advertise the feature 'urn:xmpp:payment:0#receipt' via Service Discovery (XEP-0030) [11].

[10] XEP-0082: XMPP Date and Time Profiles <https://xmpp.org/extensions/xep-0082.html>.
[11] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

5.2.6 The get-invoice Element

The <get-invoice/> element is the payload of a proactive invoice request. It is sent as the child of an IQ-get stanza and possesses the following attributes:

- service (REQUIRED) — A registered service type identifier (see Invoice Request Service Registry).
- target (OPTIONAL) — The JID of the specific resource being requested (e.g., the JID of a MUC room).

5.3 Error Flow

When a service requires payment for an action, it MUST respond to the triggering stanza with an error of type "auth" containing the <payment-required/> application condition element qualified by 'urn:xmpp:payment:0', and MUST include an <invoice/> element as a sibling of the error condition element within the <error/> element.
The "auth" error type is appropriate because the payer has not yet demonstrated authorization (via payment) to access the resource. The <payment-required/> application condition distinguishes this case from <not-authorized/> and <registration-required/>.
If the requesting entity is not authenticated at all (e.g., it has not completed SASL authentication with its own server, or its JID cannot be verified), the service SHOULD return a <not-authorized/> error rather than a <payment-required/> error. Payment challenges SHOULD only be issued to entities whose identity can be established, to prevent anonymous entities from exploiting the payment flow to probe service behavior.
The service SHOULD include a human-readable <text/> child within the <error/> element describing the reason for the payment requirement.
When a <payment/> element is present on a retried stanza but verification fails, the service SHOULD return a <not-acceptable/> error of type "modify" containing a <payment-required/> element with an appropriate 'reason' attribute (see The payment-required Element), and MAY include a new <invoice/> element to allow the payer to retry with a fresh payment. Using a machine-readable 'reason' value allows automated agents to distinguish between a recoverable failure (e.g., 'payment-expired', which warrants requesting a new invoice) and a non-recoverable one (e.g., 'verification-failed', which warrants escalating to the user).
Some payment systems settle asynchronously: the service may have detected a payment for the session but cannot yet confirm it has settled (e.g. a bank transfer awaiting reconciliation, or a cryptocurrency payment awaiting confirmations). In this case, where the service does not need a proof from the payer, only more time, it SHOULD return a stanza error of type **"wait"** (per RFC 6120 [12], signifying a transient condition that warrants retrying after a delay) containing a <payment-required/> element with reason='payment-pending', and SHOULD include a retry-after attribute giving the number of seconds the payer should wait. The payer then retries the *same* session after the indicated interval; it does not pay again. This differs from the verification-failure case both in error type ("wait" rather than "modify"/"auth")

[12] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

and in semantics: the payment is on its way, not rejected.

5.4 Verification

The mechanism by which a service verifies that a payment has been made is outside the scope of this specification. The following non-normative guidance is provided for implementors.
For bank transfers (SEPA, IBAN, UPI, PIX, SWIFT, etc.), the service typically cannot verify payment in real time. The service SHOULD encode the session identifier as the payment reference in the payto URI (the 'message' or 'instruction' query parameter) and reconcile incoming payments against outstanding sessions via its bank's API or statement-import mechanism. Because bank transfers can take from seconds (SEPA Instant) to days (SWIFT) to settle, bank transfer options are best suited to one-time access grants such as account registration or monthly subscriptions rather than per-message micropayments.
For Lightning Network payments, the service MAY verify payment by stateless preimage verification: if the payment hash is encoded in the session identifier, the service verifies that SHA-256(preimage) matches the payment hash without querying external state. Alternatively, the service MAY query its own Lightning node for a settled invoice.
A facilitator service MAY issue an opaque bearer token after verifying payment. The token can be used across multiple retried stanzas within its validity period, enabling a "pay once, use many times" pattern. The format and lifecycle of such tokens is outside the scope of this specification.
A service MUST NOT accept the same session identifier more than once. Once a session is consumed, the service MUST invalidate it. A service MUST NOT honor a session after the time specified in the 'expires' attribute.

6 Business Rules

The following rules apply to implementations of this specification.

6.1 Service Behavior

- A service MUST generate a new, unique session identifier for every invoice issued.
- A service MUST include at least one payment option.
- A service MUST NOT include personally identifiable information in invoice payloads beyond what is required for payment routing.
- Services SHOULD keep session expiry times appropriately short for instant-settlement options (10 to 30 minutes) and appropriately generous for slower bank-transfer options (1 to 3 business days).

6.2 Client Behavior

- Clients SHOULD present all payment options to the user, ordered from most to least preferred, to allow the user or automated agent to select the most suitable option.
- Automated agents SHOULD prefer instant-settlement options because these can be completed and verified programmatically without user interaction.
- Clients presenting a user interface SHOULD render URI-based options (e.g., payto URIs) as QR codes to facilitate payment from a mobile banking or wallet application.
- A client MAY treat a payment URI as an opaque string and delegate it to an external payment application (via an operating-system URI handler or a QR code) without parsing it. Such a client relies on the payment application to display the authoritative amount and beneficiary and to obtain the user's final authorization.
- A client MAY present a payment option without recognising its URI scheme by rendering the URI as a QR code for the user to scan, hand it to an operating-system URI handler, or show it as selectable text. Recognising the scheme is not a prerequisite for offering an option, and a client SHOULD NOT hide an option merely because it cannot parse its scheme. An option is genuinely unactionable only when the client can surface its URI to the payer in none of these ways (most plausibly an automated agent paying without human interaction and with no means of settling that scheme); such a client SHOULD act on whichever options it can and report that the remainder cannot be fulfilled rather than silently discarding them.
- A client that implements automated payment (paying without a human reviewing the amount in a payment application) MUST obtain the amount from the payment URI itself.
- Clients MUST NOT silently retry a stanza without presenting payment options to the user unless the client has been explicitly configured by the user to auto-pay up to a defined amount threshold.
- If the 'expires' time has passed before the user initiates payment, the client SHOULD request a fresh invoice via the proactive IQ flow rather than attempting to pay an expired one.

7 Implementation Notes

The 'payto' URI scheme (RFC 8905) is the RECOMMENDED primary URI scheme for traditional bank rails, because it covers the widest range of them within a single IETF-standardized format. Examples of valid payto URIs include:

- payto://iban/DE75512108001245126199?amount=EUR:200.00&message=hello (SEPA bank transfer)

- payto://upi/merchant@examplebank?amount=INR:200&receiver-name=Example (India UPI)
- payto://ach/122000661/0000000123?amount=USD:200.00 (US ACH transfer)

Because each option carries a payment URI, any payment system reachable through a registered URI scheme is usable without modifying this specification. Cryptocurrency networks are often better addressed through their own native schemes rather than through a payto target type because typically the native scheme is the form the payer's wallet recognises (see The option Element).
A QR code is normally generated directly from the payment URI: the client renders the URI string as a QR code without interpreting it, and the payer scans it with a wallet or banking application. This works natively for cryptocurrency URI schemes. For some bank-transfer systems, however, the payment application expects a different QR payload (for example, banking apps in several European countries scan a European Payments Council EPC069-12 payload rather than a payto URI). In that case the service MAY additionally supply that payload in the <qr/> child of the option, which the client likewise renders as a QR code without interpreting it. The payment URI itself remains REQUIRED, so a machine-readable, scheme-identified payment instruction is always present.

8 Accessibility Considerations

Clients MUST NOT present a payment interface as the sole means of completing an action where an accessibility-equivalent alternative exists. Where a service also offers a CAPTCHA Forms (XEP-0158) [13] CAPTCHA challenge, the payment option and the CAPTCHA option SHOULD be presented with equal prominence.
When rendering a payment option as a QR code, whether from the payment URI or from a <qr/> payload, clients SHOULD also present the underlying payment URI as selectable text so that users of screen readers or other assistive technologies can copy and use it directly.

9 Internationalization Considerations

The 'purpose' attribute of <invoice/> and the 'label' attribute of <option/> are human-readable strings. Services SHOULD provide these in the language of the recipient where known. Multiple language variants MAY be provided using the standard 'xml:lang' attribute on any element containing character data.
This specification does not define an amount or currency syntax of its own: the amount, currency, and any denomination are carried inside the payment URI and are governed by that URI scheme's own standard (for payto, the RFC 8905 currency:amount notation). A client that displays the amount derives it from the URI and SHOULD format it for the recipient's locale.

[13] XEP-0158: CAPTCHA Forms <https://xmpp.org/extensions/xep-0158.html>.

10 Security Considerations

The following security considerations apply to implementations of this specification.

10.1 Replay Attacks

Session identifiers MUST be unique and single-use. A service MUST maintain a record of consumed session identifiers for at least as long as the invoice expiry window to prevent replay attacks. Reusing a session identifier to obtain multiple access grants MUST be rejected.

10.2 Session Binding

Because XMPP servers routinely mutate stanzas in transit — adding elements such as <delay/>, rewriting the 'from' attribute, injecting stream management acknowledgements, and normalizing namespace prefixes — it is not possible for a service to cryptographically bind an invoice to the byte content of the stanza it declined. Body-level hashing, as used by HTTP-based payment protocols, is therefore not applicable in XMPP.

A service therefore MUST verify a session against parameters it knows itself, never against any property of the client's retry stanza; this is what allows verification to survive server-side stanza mutation. The session value MUST be unguessable and MUST be bound to what was purchased (e.g. the resource the invoice grants access to and the amount required) so that a session obtained for one resource or amount cannot be redeemed for another. Where the invoice was issued in response to a proactive <get-invoice/> request, the 'target' resource SHOULD be part of this binding.

How the service achieves this is an implementation choice. It MAY store server-side state mapping the session to the invoice, or encode the binding into a self-contained keyed token, for example an HMAC over those service-side parameters (not the option URIs, which embed the session and are not echoed on retry) together with a per-invoice nonce, so that each session stays unique and can be re-verified without storing the invoice. Either way the service MUST record consumed sessions until they expire (see Replay Attacks): the token approach avoids storing invoice parameters but not the set of consumed sessions.

10.3 Invoice Tampering

An intermediary XMPP server could modify <invoice/> contents in transit, redirecting payment destinations to attacker-controlled accounts. Implementations SHOULD use end-to-end encryption (e.g., OMEMO Encryption (XEP-0384) [14]) when the integrity of invoice contents is critical. For Lightning Network options, stateless preimage verification protects the service from granting access without payment: because the session identifier is bound to the payment hash (see Session Binding), a tampered invoice would yield a preimage that

[14] XEP-0384: OMEMO Encryption <https://xmpp.org/extensions/xep-0384.html>.

fails verification. This does not protect the payer, who may still send funds to an attacker; end-to-end encryption is required for that.
For traditional bank transfer options, no equivalent cryptographic binding between the invoice and the beneficiary account exists. Users SHOULD independently verify beneficiary account details before initiating a bank transfer, particularly when communicating with a service for the first time.

10.4 Amount Presentation Integrity

The authoritative amount and beneficiary of a payment are those encoded in the payment URI of the chosen <option/>. This specification deliberately carries no separate human-readable amount field: a duplicate amount supplied alongside the URI could not be verified against it by a client that does not parse the scheme, and a preview diverging from the URI is a spoofing risk. The label and purpose attributes remain human-readable text describing the option and the reason for payment.
Accordingly:

- A client that displays an amount MUST derive it from the payment URI, and MUST NOT treat the label, purpose, or any other free-text field as an amount or use such a field as the basis for an automated decision, including the financial-safety threshold check of Financial Safety.
- A client is not required to parse payment URIs. A client that treats the URI as opaque and delegates it to a payment application relies on that application to display the authoritative amount and beneficiary and to obtain the user's final authorization; such a client SHOULD make clear that the binding amount and beneficiary are those confirmed in the payment application. This is the established "what you see is what you sign" model: the trusted endpoint that authorizes the payment is also the one that re-displays its details.
- A client implementing automated payment (with no human reviewing the amount in a payment application) MUST derive the amount from the URI itself, since there is no downstream party to verify it.
- A <qr/> payload, where present, is a rendering convenience and carries the same in-transit tampering risk as the URI; it is not a substitute for the payment application's confirmation of the authoritative amount and beneficiary. A client able to interpret both the URI and the <qr/> payload SHOULD verify that they describe the same payment.

End-to-end encryption (e.g. OMEMO Encryption (XEP-0384) [15]) protects the integrity of the URI in transit.

[15] XEP-0384: OMEMO Encryption <https://xmpp.org/extensions/xep-0384.html>.

10.5 Financial Safety

Clients MUST NOT automatically pay an invoice above a configurable amount threshold without explicit user confirmation. Implementations SHOULD default this threshold to zero (i.e., all payments require explicit user approval) and SHOULD allow the user to raise it. Automated agents operating within a pre-authorized budget MAY raise this threshold programmatically for their specific use case, but MUST NOT do so without the knowledge and consent of the account holder.

11 Privacy Considerations

Services MUST NOT include information in invoice metadata that would allow correlation of payments to real-world identities beyond what is required for the service function.
Lightning Network payment hashes are pseudonymous. However, a service that retains proof-of-payment preimages alongside session records can link a payment to the payer's JID. Services SHOULD minimize the identity information stored alongside consumed session records and SHOULD delete such records once the access grant has expired.
Traditional bank transfer options inherently reveal the payer's real name and account details to the payee, as this is a property of the underlying banking system. Clients SHOULD inform the user of this before initiating a bank transfer to a party the user has not previously transacted with.
The 'session' value SHOULD NOT encode any information about the payer's identity or behavior.

12 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA).

13 XMPP Registrar Considerations

13.1 Protocol Namespaces

The XMPP Registrar [16] shall include 'urn:xmpp:payment:0' in its registry of protocol namespaces.
If the protocol defined in this specification undergoes a revision that is not fully backwards-compatible with an older version, the XMPP Registrar shall increment the protocol version number found at the end of the XML namespaces defined herein, as described in Section 4 of

[16] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

XEP-0053.

13.2 Service Discovery Features

The XMPP Registrar [17] shall include the following features in its registry of service discovery features (see <https://xmpp.org/registrar/disco-features.html>):

- urn:xmpp:payment:0
- urn:xmpp:payment:0#invoice-request
- urn:xmpp:payment:0#receipt

An entity that supports the error-flow portion of this protocol MUST advertise the feature 'urn:xmpp:payment:0'. An entity that additionally supports the proactive IQ-based invoice request MUST advertise 'urn:xmpp:payment:0#invoice-request'. An entity that supports emitting <receipt/> elements upon successful payment MUST advertise 'urn:xmpp:payment:0#receipt'.

13.3 Payment Schemes and Proof Types

This specification deliberately does **not** establish an XMPP Registrar [18] registry of payment schemes, payment systems, or proof-of-payment formats. Curating payment systems is outside the XSF's area of expertise and is already handled by external bodies. Payment options are identified by their URI scheme, ideally via the IANA URI Schemes registry, and for the payto scheme by the payment target types maintained externally under RFC 8905. Proof types are declared per option by the issuing service and echoed by the payer (see The payment Element). The conventional values listed there are non-normative and their formats are defined by the relevant payment systems' own specifications.

13.4 Invoice Request Service Registry

This is the one registry this specification defines, and it concerns XMPP application contexts (a domain within the XSF's remit) rather than payment systems.

[17] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

[18] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

The XMPP Registrar [19] shall maintain a registry of service type identifiers for use in the 'service' attribute of the <get-invoice/> element. The initial contents of this registry are as follows.

Value	Description
muc-entry	Entry fee for a Multi-User Chat (XEP-0045) XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>. room.
file-upload	Fee for a file upload service (e.g., HTTP File Upload (XEP-0363) XEP-0363: HTTP File Upload <https://xmpp.org/extensions/xep-0363.html>.).
message	Per-message fee, e.g., for a bot service or an anti-spam deposit.
registration	Account registration or hosting fee.

14 Design Considerations

Several alternative designs were considered during the development of this specification.

14.1 Using Data Forms for the Invoice

Data Forms (XEP-0004) [20] Data Forms were considered as the vehicle for invoice delivery, since CAPTCHA Forms (XEP-0158) [21] already uses them for challenges. Data Forms were rejected for the primary invoice format because the structured multi-element nature of an invoice (multiple <option/> children, each with a payload string and a display annotation) does not map naturally onto flat form fields, and because a dedicated element is more self-describing and easier to parse for automated agents. Data Forms are retained only for the CAPTCHA Forms (XEP-0158) [22] integration use case, where the payment option appears as a single field alongside other challenge fields.

14.2 Choosing auth as the Error Type

The "auth" error type was chosen for the payment-required error rather than, say, "cancel" or a new type, because the payer genuinely lacks authorization to perform the requested action until payment is made. This mirrors the semantics of HTTP 402 and is consistent with the existing use of "auth" for <registration-required/> in RFC 6120 [23].

[19] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.
[20] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.
[21] XEP-0158: CAPTCHA Forms <https://xmpp.org/extensions/xep-0158.html>.
[22] XEP-0158: CAPTCHA Forms <https://xmpp.org/extensions/xep-0158.html>.
[23] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

14.3 Streaming Payment Sessions

This specification defines only single-payment (”charge”) semantics: one invoice, one payment, one grant. For high-frequency use cases such as per-query AI agent bots, this model requires a new invoice round-trip for every request, which introduces latency and overhead. A future companion specification MAY define streaming payment session semantics, analogous to the session intent in the Machine Payments Protocol (MPP). In such a model, an agent would open a payment channel by pre-funding it, and subsequent requests within the same channel would be settled using off-chain cryptographic vouchers without requiring a new invoice or an on-chain transaction per query. Such a specification would advertise support via a new feature (e.g., ’urn:xmpp:payment:0#session’) and is outside the scope of this document.

15 XML Schema

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    elementFormDefault='qualified'
    targetNamespace='urn:xmpp:payment:0'
    xmlns='urn:xmpp:payment:0'>

  <!-- invoice: emitted by services in error stanzas and IQ results --
    >
  <xs:element name='invoice'>
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs='1'
                    minOccurs='0'
                    ref='description'/>
        <xs:element maxOccurs='unbounded'
                    minOccurs='1'
                    ref='option'/>
      </xs:sequence>
      <xs:attribute name='expires'
                    type='xs:dateTime'
                    use='optional'/>
      <xs:attribute name='purpose'
                    type='xs:string'
                    use='optional'/>
      <xs:attribute name='session'
                    type='xs:string'
                    use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='description' type='xs:string'/>
```

```
<!-- option: one payment option (a payment URI) within an invoice --
    >
<xs:element name='option'>
  <xs:complexType mixed='true'>
    <xs:sequence>
      <xs:element maxOccurs='1'
                  minOccurs='0'
                  ref='qr'/>
    </xs:sequence>
    <xs:attribute name='label'
                  type='xs:string'
                  use='optional'/>
    <xs:attribute name='proof'
                  type='xs:string'
                  use='optional'/>
  </xs:complexType>
</xs:element>

<!-- qr: a payment string to be rendered as a QR code, for use when
    the URI is not the scannable payload -->
<xs:element name='qr' type='xs:string'/>

<!-- payment: added to retried stanzas by the payer -->
<xs:element name='payment'>
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs='1'
                  minOccurs='0'
                  ref='proof'/>
    </xs:sequence>
    <xs:attribute name='session'
                  type='xs:string'
                  use='required'/>
  </xs:complexType>
</xs:element>

<xs:element name='proof'>
  <xs:complexType mixed='true'>
    <xs:attribute name='type'
                  type='xs:string'
                  use='required'/>
  </xs:complexType>
</xs:element>

<!-- payment-required: application-specific stanza error condition
    -->
<xs:element name='payment-required'>
  <xs:complexType>
```

```
      <xs:attribute name='reason'
                    type='xs:string'
                    use='optional'/>
      <xs:attribute name='retry-after'
                    type='xs:nonNegativeInteger'
                    use='optional'/>
    </xs:complexType>
  </xs:element>

  <!-- receipt: optional element returned by the service upon
      successful settlement -->
  <xs:element name='receipt'>
    <xs:complexType>
      <xs:attribute name='reference'
                    type='xs:string'
                    use='optional'/>
      <xs:attribute name='session'
                    type='xs:string'
                    use='required'/>
      <xs:attribute name='settled'
                    type='xs:dateTime'
                    use='optional'/>
    </xs:complexType>
  </xs:element>

  <!-- get-invoice: IQ-get payload for proactive invoice request -->
  <xs:element name='get-invoice'>
    <xs:complexType>
      <xs:attribute name='service'
                    type='xs:string'
                    use='required'/>
      <xs:attribute name='target'
                    type='xs:string'
                    use='optional'/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```