Abstract

DIX is an Internet scale protocol for exchanging identity information between endpoints. The protocol architecture maintains a separation of control between all parties of the exchange and supports both compartmentalized and anonymous identities.
Conventions used in this document

In examples, "C:" and "S:" indicate lines sent by the client and server respectively.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

Table of Contents

1. Introduction...................................................3
2. Definitions....................................................3
  2.1. User......................................................3
  2.2. Identity Data................................................3
  2.3. Persona.....................................................3
  2.4. Homesite..................................................3
  2.5. Membersite................................................3
  2.6. Dumb Client...............................................4
3. Overview.....................................................4
  3.1. Transport Bindings..........................................4
  3.2. Names and Types.............................................4
    3.2.1. URLs and URIs..........................................5
    3.2.2. Site Paths.............................................5
    3.2.3. Site Endpoint URL......................................5
    3.2.4. Persona URL...........................................6
    3.2.5. Site Document..........................................6
    3.2.6. DIX URI Namespace.....................................6
4. Information Model..............................................7
  4.1. Definitions..............................................7
  4.2. Properties.................................................7
5. Capabilities...................................................8
  5.1. Capability Definition......................................8
  5.2. Capability Families.......................................8
  5.3. Capability URI.............................................8
  5.4. Service Capability.........................................9
  5.5. Property Capability.......................................9
  5.6. Capability Discovery and Negotiation......................9
  5.7. Capability Resolution.....................................9
  5.8. Capability Revisions......................................10
  5.9. Defined Capability Families............................10
    5.9.1. DIX Capabilities.....................................10
  5.10. Protocol................................................10
    5.10.1. Discovery Process..................................11
    5.10.2. Exchange Process..................................14
    5.10.3. Verification Process..............................24
1. Introduction

This document specifies a protocol for the exchange of identity information.

2. Definitions

2.1. User

A person with a digital identity who participates in DIX based identity information exchanges using their client software, typically a web browser.

2.2. Identity Data

A property of a digital identity in which the Property Name and Property Value are represented as a name-value pair.

2.3. Persona

A user can have multiple personas as part of their identity. For example, a user might have a work persona and a home persona. A persona is a subset of the user’s identity data.

2.4. Homesite

The User’s agent (either a website or an application) responsible for authenticating and identifying the user, providing a repository for their identity data, and releasing that data (with user consent) to other sites via the user’s client.
2.5. Membersite

A site that requests identity data from a Homesite via the user’s client.

2.6. Dumb Client

A regular web browser without any knowledge of the DIX protocol.

3. Overview

The core of the DIX protocol is a set of procedures for discovery of a Homesite, exchange of identity information, and verification of that exchange. Without drilling into too many details, this is a general description of the DIX protocol.

The User browses to a Membersite. The Membersite discovers the User’s Homesite and its capabilities. The Membersite then sends a request for identity information to the Homesite through the User’s client. The Homesite processes the request, prompting the User for consent and returns a response, again through the User’s client. The Membersite then checks integrity of the response and verifies it with the Homesite.

3.1. Transport Bindings

The protocol messages could be moved over many alternative transport mechanisms. For example, HTTP, SMTP, or XMPP. Even within a single transport layer messages could be moved in alternative ways.

This document only details a binding for HTTP. The messages are moved via HTTP POST. [RFC2616]

For the simple usage profile discussed here we’re interested in moving messages over HTTP(S). In this document, wherever HTTP is mentioned HTTPS can be used as an alternative to provide greater confidentiality and integrity.

3.2. Names and Types

Some aspects of the protocol, the names and types of things, are important enough that they need to be discussed ahead of the protocol.
3.2.1. URLs and URIs

Throughout this protocol specification there are many things of type URI. These must all be normalized to ensure they can be easily compared. RFC 2396bis describes URI normalization. [RFC2396BIS]

3.2.2. Site Paths

A Membersite or Homesite is referred to via its Path, which is a domain name and path. Since the Path is anchored within the domain name system we are assured that the name has the property of being unique.

3.2.2.1. Membersite Path

This is the unique identifier for the Membersite, and must contain a domain name. The Homesite uses this to provide a more seamless experience to the user on subsequent visits to the Membersite, and is the ID used in Membersite identity verification.

For example: membersite.com/, or membersite.com/shopping/.

3.2.2.2. Homesite Path

A unique name for a Homesite that a user can remember so they can type it in. The Homesite Path is used to discover the Endpoint URL and Capabilities of a Homesite.

For convenience, when a user presents their Homesite Path, the software should make every effort to transform the Homesite Path into a canonical URL. Termed the Homesite URL. For example, the user should be able to type in google.com, amazon.com, yahoo.com, sxip.com, ford.com, or whatever and have the system discover the rest.

For example: homesite.com/, or homesite.com/~dick/.

3.2.3. Site Endpoint URL

Membersites and Homesites provide endpoints for protocol messages to be sent to. These are specified as URLs, and over HTTP the messages are received via HTTP POST

3.2.3.1. Membersite Endpoint URL

A Membersite may have multiple Endpoint URLs, as such the Membersite Endpoint URL must not be used as a unique identifier for a
Membersite. The Membersite Endpoint URL must contain the Membersite Path.

3.2.3.2. Homesite Endpoint URL

As well as specifying an endpoint, this is used as a unique identifier for the Homesite.

3.2.4. Persona URL

A Persona URL is the identifier for a persona. In the simple profile the Persona URL is a URL that can be resolved to a web page owned by the User. We term this the Persona URL.

3.2.5. Site Document

The Homesite URL and the Persona URL reference Documents that contain marked up data for inspection by Membersites.

3.2.5.1. Persona Document

The Persona Document delegates authority for the Persona URL to a Homesite. [See Delegation Tag]

3.2.5.2. Homesite Document

The Homesite Document advertises the Endpoint URL and Capabilities of the Homesite. [See Homesite Tag]

3.2.6. DIX URI Namespace

To ensure protocol extensibility we reuse the same naming scheme throughout the protocol for multiple purposes. All names are URIs. The generic URI syntax is as follows.

uri = scheme ":" hier-part [ "?" query ] [ "#" fragment ]

hier-part = "//" authority path-abempty

/ path-absolute

/ path-rootless

/ path-empty
For a DIX URI, the protocol scheme used is 'dix'. (Note that this has not been registered with IANA, but we will resolve this during the IETF standardization process.)

dix-uri = "dix" ":" hier-part [ "?" query ] [ "#" fragment ]

Extensibility stems from the authority. Anyone with a registered domain name can create DIX URI Names using their own domain name as the authority in the URI.

The DIX protocol uses DIX URI Names for:

- Capability Names
- Property References
- Message Parameters
- Constant Values

4. Information Model

4.1. Definitions

A homesite has a set of user accounts.

An account has a set of personas.

A persona has a set of properties.

A property has a property name and property value.

A property name is a path.

A property value is a UTF-8 string.

A path is of the form 1*( "/" 1*ALPHA ). For example: '/sxip.net/contact/name/first'.

4.2. Properties

The property name is used for both referring to a property in a query expression [See Fetch Request Message], and for referring to the associated type information. [See Fetch Request][See Capabilities]
5. Capabilities

5.1. Capability Definition

A capability is a service or a property: a service offered by a site to other sites, or a property supported by a site. Sites advertise their capabilities to enable capability discovery and negotiation with other sites.

5.2. Capability Families

A capability family is a set of capabilities. The capability family mechanism is provided to reduce the number of capabilities that a site needs to advertise. Otherwise, over time, as the number of capabilities supported increases, the number of capabilities advertised could become unmanageable; the solution offered here is to roll a set of them into a family.

5.3. Capability URI

A capability is represented as a Capability URI, which is a DIX URI Name, with some restrictions.

```
capability_uri = "dix:/" [ "/" authority "/" ] capability "#"
revision

capability = 1*ALPHA
revision = 1*DIGIT
```

The Capability URI is restricted in that after the optional authority, the path contains a single step, which is the capability name, and always includes a fragment, which is the capability revision number.

The optional authority denotes whether the capability is a DIX, or Non-DIX capability. DIX capabilities have no domain name. For example, ‘dix:/core#1’. Non-DIX capabilities do have a domain name. For example, ‘dix://sxip.net/simple#1’.

The revision number encoded as the URI fragment allows a recipient to determine if a particular revision of a capability is supported. [See Capability Revisions.]
5.4. Service Capability

Service capabilities represent a service provided by the site. Example: dix://crypto-doodes.com/dongle#5’. (This was invented for this example, but hopefully you get the idea.)

5.5. Property Capability

Property capabilities represent a property supported by a site. Example: dix://sxip.net/namePerson/first

5.6. Capability Discovery and Negotiation

Homesites advertise their capabilities in the Homesite Tag of their Homesite Document. [See Homesite Document][See Homesite Tag]. Membersites read this document to discover the capabilities of the Homesite. A Membersite treats the Homesite capabilities as a hint of the services offered and the properties supported.

Membersite may provide its capabilities to a Homesite within a request message. The Homesite treats the Membersite capabilities as a hint.

5.7. Capability Resolution

A Capability URI can be converted into a resolvable URL. The document fragment referenced describes the capability. There are two anticipated use cases for resolving capabilities. Firstly, it is expected that programming tools will resolve capabilities to aid programmers in development of sites. Secondly, Homesites could use this mechanism for automatic update of capability revisions.

The Capability URI is converted to a URL by replacing the ‘dix’ protocol scheme with the ‘http’ protocol scheme. The document fragment retrieved contains a list of capabilities. For example, the capability URI ‘dix://sxip.net/simple#1’ can be converted to the URL ‘http://sxip.net/simple#1’ and resolved.

In the case of a DIX capability, where there is no domain name provided, the domain name used is a well-known, standards body name. Currently it is ‘dixs.org’. So, for example, the DIX capability ‘dix:/core#1’, would become the resolvable URL, ‘http://dixs.org/core#1’.

Note that ‘dixs.org’ is currently operated by Sxip Identity. Once DIX has progressed further within the IETF it is anticipated that this
resolution service will most likely be changed to some well-known community owned naming registry.

The URL references a document fragment. That schema of that fragment is not currently defined.

5.8. Capability Revisions

A capability revision is a positive integer value that always increases when there is a revision of a capability; as such capabilities are always backwardly compatible. This allows capability definitions to be accretive so functionality can be extended without breaking existing functionality.

For example, the capability URL ‘http://sxip.net/simple#34’ would reference where revision ‘34’ of the ‘simple’ capability is defined. From that point in the document onwards is what is included in ‘dix://sxip.net/simple#34’.

Capability revisions are always added to the top of the capability definition file, so that everything from that fragment onwards is what defines that revision of the capability.

The revision fragment allows families of capabilities to be rapidly extended without breaking existing applications.

5.9. Defined Capability Families

The following DIX capability families are currently defined. A DIX capability family is represented by a DIX URI. Third-party authorities are able to define their own capability families if they wish.

5.9.1. DIX Capabilities

5.9.1.1. dix:/core#1

Core DIX protocol capabilities. The simple usage profile. Our assumptions are: low security, low risk identity transaction, data is moving over HTTP(S), low risk from replay attack. Supports: Homesite, Membersite, and Persona URL.

5.10. Protocol

In the simple usage profile the participants in the protocol are: the User's Client, a Homesite, a Membersite, and the site that hosts the Persona URL.
The protocol proceeds in three stages: Discovery, Exchange, and Verification. We walk through each in turn.

5.10.1. Discovery Process

The goal of the discovery process is to determine the capabilities of the user’s Homesite.

A dumb client can use a cookie.

5.10.1.1. Membersite Cookie (Dumb Client)

Once a user has visited a Membersite and identified the name of their Homesite that Membersite can cookie the user with their choice of Homesite, so that it can be filled in automatically for them in the future.

Also, the name of the text control used by the Membersite to collect the user’s Homesite Path should be the consistent and well-known name ‘dix:/homesite’. The name of the text control is defined to ensure the benefit to the user of a browser’s automated form filling functionality.

5.10.1.2. User Prompt

The Membersite first attempts to determine the Homesite Path from a Membersite cookie. It then returns a web page to the user’s client. The web page contains a form that includes a text control named ‘dix:/homesite’ for the Homesite Path and an adjacent ‘login’ button. If known the Homsite Path is provided as the value of the text control. The form also includes the parameters for a fetch-request message. The parameters include the request message parameters and queries for any identity data the Membersite requires. [See Request Message for details.]

The attributes of the FORM tag are:

- CLASS attribute must contain ‘DIX’, which denotes to a smart client that the form was generated by a DIX aware Membersite.
- ACTION attribute is the Membersite Endpoint URL.
- METHOD attribute is ‘POST’.

The value of the FORM tag includes at least two INPUT tags, one a text control, the other a button control.
The attributes of the text control INPUT tag are:

- TYPE attribute is 'TEXT'.
- NAME attribute is 'dix:/homesite'.
- VALUE attribute is the user’s Homesite Path, if known by the Membersite.

The attribute of the button control INPUT tag are:

- TYPE attribute is 'SUBMIT'.
- VALUE attribute is 'login'.

The attributes of the hidden INPUT tags for the message parameters are:

- TYPE attribute is 'HIDDEN'.
- NAME attribute is the message parameter name.
- VALUE attribute is the message parameter value.

Here’s an example:

```html
<form class="DIX" method=POST action= "http://membersite.com/sxip/homesite_discovery">
  <input type="hidden" name="fname" value="dix://sxip.net/contact/name/first"/>
  <input type="text" name="dix:/homesite" value=""/>
  <input type="submit" value="login"/>
</form>
```

Processing of the form by the client differs between smart and dumb clients.
5.10.1.3. Dumb Client Form Processing

When a dumb client receives a page containing this form the user is presented with a text control with an adjacent ‘sxip in’ button. If the user has previously visited the Membersite, and the Membersite has stored the user’s Homesite Path in a cookie, then the Membersite can suggest a value for the text control. Otherwise, the user will have to type in their Homesite Path, and potentially their browser will auto-complete it after a few key strokes.

When the user clicks on the ‘sxip in’ button the form is posted to the Membersite Endpoint URL. The Membersite should accept a shortened form of URL from the user and make every effort to transform it into a canonical URL. The Membersite then performs the Homesite Inspection process.

5.10.1.4. Homesite Inspection

The Membersite must now determine the Endpoint URL and the Capabilities of the User’s Homesite. The Membersite uses the Homesite Tag Inspection Algorithm to discover the Homesite Tag.

The Membersite uses the Homesite URL to fetch the Homesite Document to inspect it for the Homesite Tag to discover the Endpoint URL and Capabilities of the Homesite.

The Membersite constructs a URL, based on the Homesite URL, to the well-known file ‘dix.html’ in the site’s root directory. If the file is not found then the Membersite resolves the URL of the site’s root document, ‘/’. Note that the Membersite must follow any redirects. If found the document retrieved is examined for the Homesite Tag.

The Homesite Tag is used to specify the Endpoint URL and Capabilities of a Homesite. The HTML LINK tag is used for this. The LINK tag is appropriate because it is a document reference that is not to be traversed by an end user. The requirements are:

- Use the LINK tag, which MUST occur in the HEAD section of the returned document.
- The LINK REL attribute MUST have value ‘dix:/homesite’.
- The LINK HREF attribute MUST have value of the Homesite’s Endpoint URL.
The LINK CLASS attribute contains the capabilities of the Homesite, space separated.

Here’s an example

```html
<Link REL="dix:/homesite" HREF="http://www.sxip.net/homesite" CLASS="dix:/core#1 dix://sxip.net/simple#1"/>
```

The reason there are options for where the Homesite Tag is located is that the User may or may not have access rights to one of these files.

5.10.1.5. Membersite’s Discovery Algorithm

To discover the Homesite Endpoint URL and Capabilities the Membersite follows the following procedure.

1. Prompt user to enter the Homesite Path.
2. Look in Homesite root for dix.html and extract Endpoint URL and capabilities.
3. If not found, then look in Homesite root page and extract Endpoint URL and capabilities.

Upon failure the Membersite should inform the user and jump to step 1 to prompt the user.

If successful the Membersite has now determined the Endpoint URL and capabilities of the user’s Homesite.

The Membersite uses the Homesite capabilities to determine if the Homesite supports the requisite capabilities for the identity data exchange. For example, a financial Membersite might require that the Homesite use a specific two-factor device to authenticate the User. In this case the Membersite would check that the Homesite capabilities discovered included the capability named ‘dix://crypto-doodes.com/dongle#5’. (Invented for this example.)

5.10.2. Exchange Process

The exchange process occurs once the MS has discovered the Endpoint URL and the capabilities of the Homesite. Messages are exchanged between the Membersite and the Homesite via the User’s client. This section describes the protocol flow that occurs and the messages that are transferred between the participants.
5.10.2.1. Protocol Flow

To set the context for the following description, the user’s client sends a HTTP GET to a Membersite, and the Membersite performs the Homesite discovery procedure.

The Membersite sends a fetch-request message to the Homesite through the User’s client via a redirected HTTP POST to their Homesite Endpoint URL using JavaScript to autosubmit the form.

The Homesite checks the integrity of the fetch-request message by checking that the Membersite Endpoint URL contains the Membersite Path.

The Homesite creates a fetch-response message. A Signature is included that is generated from a Digest of the message.

Membersite receives the fetch-response message and verifies the message.

5.10.2.2. Digest Generation

The digest function is used to generate a digest for a message. It takes as input a canonicalized representation of the message.

\[
\text{Digest} = D \left( C \left( M \right) \right)
\]

Where, \( M \) is the fetch-response message excluding the ‘\text{signature}\’ message parameter, \( C \) is the canonicalization function (defined below), and \( D \) is the digest function.

Since the Homesite and Membersite must be able to generate interchangeable digests they both must implement the same digest function. We specify that both must implement the SHA-1 algorithm. [SHA]

5.10.2.3. Signature Generation

The signature function is used to generate a signature for a message. Since the signature generation function need only be known by the Homesite, and not the Membersite, the nature of this function is implementation defined. It should however have the properties of protecting the message from alteration.

A suggested implementation of a signature function would be to use the SHA1 algorithm, which takes as input a digest of the message and a secret known only to the Homesite.
Signature = T ( S + Digest )

Where, Digest is message digest (defined above), S is the Homesite Secret, T is the signature generation function, and ‘+’ means string concatenation.

A suggested secret would be a random sequence of bytes. For the SHA1 algorithm an appropriate length would be 80 to 160 bits.

5.10.2.4. FORM Auto Submission

Form submission could be automated in the following way:

```html
<html>
<body onload="document.theForm.submit()">
<form name="theForm" method="post" action="http://membersite.com/url">
  <input type="hidden" name="dix:/message-type" value="dix:/fetch-response" />
  .
  .
  .
  etc
  
  <input type="submit" value="Post" />
</form>
</body>
<noscript>
<p>Click "Post" to complete the transaction.</p>
<p style="font-size: small">Note: you are only seeing this page because you have Javascript disabled, or because your browser does not support Javascript.</p>
</noscript>
```
5.10.2.5. Messages

A message is a list of name-value pairs, which are represented here as ‘<name>=<value>’. A name or value could begin with a well-known escape sequence that denotes that the thing is protocol oriented and that extra processing is required. The escape sequence is the character sequence ‘dix://’.

The escaped thing is extensible via a namespace mechanism. If the escape sequence is followed by another forward slash character, ‘/’, then the following domain name denotes the namespace within which the following name belongs. For example, a name in the ‘sxip.net’ domain would begin with the sequence, ‘dix://sxip.net’, so the name ‘foo’ would be ‘dix://sxip.net/foo’.

In a message there are three possible forms of a name-value pair:

5.10.2.5.1. Message Parameter

Request: dix://<name>=<value>

Response: dix://<name>=<value>

Description: Protocol specific. Requires processing by the Membersite or Homesite.

5.10.2.5.2. Data Request / Response

Request: <name>=dix://<query>

Response: <name><value>

Description: Homesite processes query and returns resultant value

5.10.2.5.3. Data

Request: <name><value>

Response: <name><value>

Description: Data is passed through. This could be used for session tracking.
5.10.2.5.4. Grammar

The above is rewritten as a formal grammar below:

Parameter = MessageParameter
  / DataRequest
  / Data

MessageParameter = Escape Name "=" Value

DataRequest = Name "=" Escape Query

Data = Name "=" Value

Escape = "dix:/"

Query = Path

Path = 1*( "/" Name )

Name = 1*ALPHA

Value = *CHAR

5.10.2.6. Messages over HTTP POST

Protocol messages are passed over HTTP as a HTML FORM that is POSTed to the recipient’s endpoint URL.

For example:

<form
  class="DIX"
  method="POST"
  action= "http://isp.com/sxip/homesite_discovery"
>
  <input type="hidden" name="dix:/required" value="fname" />
  <input type="hidden" name="fname" value="dix://sxip.net/contact/name/first"/>
</form>
5.10.2.7. Message Canonicalization

This algorithm performs message canonicalization before computation of a digest, by default with the SHA-1 algorithm.

For each name-value pair within the message, with the exception of the signature parameter, treat as a series of bytes, remove any trailing linefeed or carriage return characters. Note that the ‘=’ character between the name and value is retained. If message parameter names or values are url-encoded, they must be decoded, most web frameworks do this automatically. Then sort the name value-pairs in byte value order and concatenate them for passing through the digest function.

5.10.2.8. Fetch Request Message Parameters

A fetch request message is sent from the Membersite to the Homesite, via the User’s Client. Its purpose is to request that the Homesite authenticate the user and to return the property values of the properties requested. Note that a fetch request need not actually request any attribute values, its purpose then is just to authenticate the user.

\[\text{dix:/message-type, required: } \text{’dix:/fetch-request’}\]

\[\text{dix:/message-datetime, optional: A UTC datetime. The Homesite can use this as a simple way to detect a message replay security attack. The Homesite would have a policy about how old a message is acceptable. For example, a couple of minutes. This assumes roughly synchronized clocks at the sites, which is a deployment issue.}\]

\[\text{dix:/message-id, optional: A nonce. [RFC2617]}\]

Optional for support of static forms and dumb Membersites who can’t generate a nonce.

An implementation of a Membersite could set to a high resolution date/time stamp. Because the data is passing over HTTP any more effort on replay attack prevention is useless.
dix:/membersite-url, required: The Membersite Endpoint URL for the Homesite to POST the fetch-response message to. This URL must include the Membersite Path, otherwise the message is invalid. Note that the URL could include return data. The code at this URL is expected to perform the Verification Process.

dix:/membersite-path, required: The Membersite Path. The Membersite Endpoint URL must include the Membersite Path, otherwise the message is invalid.

dix:/membersite-name, recommended: The display name for the Membersite.

dix:/signature, optional: The Membersite provides the Homesite with a signature, so that the Homesite can directly verify with the Membersite that the request message did come from the Membersite. [See Verification Process for details.] The value of the name-value pair is encoded as lower-case hex without any leading characters, such as ‘0x’.

dix:/membersite-accept, recommended: The Membersite can optionally provide the capabilities that it supports. The Homesite can then tailor its response to the appropriate capability revisions.

dix:/authentication-age, optional: An integer value, which is a number of seconds. If the user has not authenticated with the Homesite since (current time – seconds) then the Homesite must re-authenticate the user. A value of 0 is taken to mean that the Homesite must always re-authenticate the user.

dix:/membersite-explanation, optional: A textual description of why the Membersite is making the request. Intended for display to the user by the Homesite.

dix:/membersite-cancel-url, optional: The value URL the Homesite should return the user to should they cancel the operation.

dix:/required, optional: The Membersite can specify that a requested property is required. This means that the Homesite should inform the user that the property is required. There is expected to be a corresponding query for the property. For example:

dix:/required=fname

fname= dix://sxip.net/contact/name/first
dix:/if-available, optional: The Membersite can specify that the Homesite should only prompt the user for the property if the property is available. For example:

dix:/if-available=fname

fname= dix://sxip.net/contact/name/first

$name=$dix:/persona-url, optional: If present, the Membersite is requesting the Persona URL. This is the unique identifier for the user’s selected persona. The Homesite must authenticate the user. The Membersite need not ask for the Persona URL if it only requires some identity data for filling in a form.

$name=$dix:/multiple-personal-url, optional: TBD

$name=$dix:/<property name>, optional: A request for a property value. The property is referenced by its property name and its value is returned assigned to the name provided in the request. For example:

fname=dix://sxip.net/contact/name/first
lname=dix://sxip.net/contact/name/last

$name=$<value>, optional: Pass through data. For example, session=F39E-4810-B3C8-B237

5.10.2.9. Fetch Response Message Parameters

Sent by the Homesite to the Membersite, via the User’s Client, in response to a Fetch Request Message.

dix:/message-type, required: ‘dix:/fetch-response’

dix:/message-datetime, required: A UTC datetime. The Membersite uses this as a simple way to detect a message replay security attack. The Membersite would have a policy about how old a message is acceptable. For example, a couple of minutes. This assumes roughly synchronized clocks at the sites, which is a deployment issue.

dix:/message-id, required: The message-id if provided in the request.

dix:/signature, optional: The Homesite sends a signature to the Membersite for use in verifying the response message.
dix:/homesite-url, required: The Homesite Endpoint URL. This is the endpoint the Membersite can use for verifying the response message with the Homesite and also can be used by the Membersite as a unique identifier for Homesite.

dix:/homesite-name, optional: A textual name for the Homesite. Intended for display to the user.

dix:/status-success, required: Status code. ‘dix:/true’ is returned if the Homesite has successfully authenticated the user. ‘dix:/false’ if the user could not be authenticated. And, ‘dix:/unknown’ if an error of some kind occurred.

dix:/status-message, optional: Human readable status message.

<name>=<persona url>, if requested: The Persona URL. The response from the request parameter ‘<name>=dix:/persona-url’.

<name>=<value>, if requested: The value is a property value for a requested property.

5.10.2.10. Membersite’s Fetch Exchange Algorithm

1. Create fetch request message.

2. Store message state. State management by the Membersite is implementation dependent. If dix:/message-id is provided then the Membersite should store the nonce to ensure that a message has not been replayed. For example, in a cookie, in a secure way.

3. Send fetch request message.


5. Verify fetch response message.

5.10.2.11. Homesite’s Fetch Exchange Algorithm

1. Receive fetch request message.

2. Verify fetch request message.

3. User interaction.

4. Create fetch response message, creating the ‘dix:/signature’ message parameter.
5. Send fetch response message.

5.10.2.12. Store Request Message Parameters

Sent from the Membersite to the Homesite, via the User’s Client.

dix:/message-type, required: ‘dix:/store-request’
dix:/message-datetime, optional: As fetch request message.
dix:/message-id, optional: As fetch request message.
dix:/membersite-url, required: As fetch request message.
dix:/membersite-path, required: As fetch request message.
dix:/membersite-name, recommended: As fetch request message.
dix:/signature, required: As fetch request message.
dix:/membersite-accept, recommended: As fetch request message.
dix:/membersite-cancel-url, optional: As fetch request message.
dix:/persona-url=value, optional: As the Membersite most likely knows
the Persona URL of the user the Membersite can optionally provide
that to the Homesite so that the identity data can be stored with
that Persona.
dix:/<property name>=<value>, optional: The identity data to store at
the Homesite.

<name>=<value>, optional: Pass through data. For example,
session=F39E-4810-B3C8-B237

5.10.2.13. Store Response Message Parameters

Sent by the Homesite to the Membersite, via the User’s Client, in
response to a Store Request Message.

dix:/message-type, required: ‘dix:/store-response’
dix:/message-datetime, required: As fetch response message.
dix:/message-id, optional: As fetch response message.
dix:/signature, optional: The Homesite optionally provides the Membersite with a signature, so that the Membersite can directly verify with the Homesite that the request message did come from the Homesite. [See Verification Process for details.]

dix:/homesite-url, required: As fetch response message.
dix:/homesite-name, optional: As fetch response message.
dix:/status-success, optional: As fetch response message.
dix:/status-message, optional: As fetch response message.

5.10.2.14. Membersite’s Store Exchange Algorithm

1. Create store request message.

2. Store message state. State management by the Membersite is implementation dependent. If dix:/message-id provided then the Membersite should store the nonce to ensure that a message has not been replayed. For example, in a cookie, in a secure way.

3. Send store request message.


5. Verify store response message.

5.10.2.15. Homesite’s Store Exchange Algorithm

1. Receive store request message.

2. Verify store request message.

3. User interaction.

4. Create store response message, creating the ‘dix:/signature’ message parameter.

5. Send store response message.

5.10.3. Verification Process

The verification process occurs after the Membersite receives a response message. The purpose of the verification process is to ensure that the response has not been tampered with and was indeed sent by the Homesite the response message claims it is from.
5.10.3.1. Protocol Flow

Upon receiving the fetch-response message the Membersite checks the integrity of the message by checking the correctness of the nonce.

If a Persona URL is returned then the Membersite then checks that the Homsite URL is authoritative for the Persona URL. The Membersite achieves this by fetching the Persona Document and inspecting it for the Delegation Tag. The Delegation Tag provides a list of Homsites that are authoritative for that persona. The Homsite URL must be within that list for the Homsite to be authoritative over the persona, otherwise the fetch-response message is invalid. [See Delegation Tag section.]

To verify the integrity of the fetch-response message the Membersite constructs a verify message that is sent to the Homsite. In the verify message is the Signature from the fetch-response message, and a message Digest computed by the Membersite. [See Digest Generation.] [See Security Considerations for why this is done.]

Upon receipt of the verify-request message the Homsite performs a comparison between the Signature provided and the result of computing a signature. [See Signature Generation.] A response containing 'dix:/true' is returned if they match, and 'dix:/false' if not. [See Security Considerations for why this is done.]

5.10.3.2. Delegation Tag

The delegation tag is used to specify the set of Homsites that are authoritative for a persona. The HTML LINK tag is used for this. The LINK tag is appropriate because it is a document reference that is not to be traversed by an end user. There could be many of these delegation tags. The requirements are:

- Use the LINK tag, which MUST occur in the HEAD section of the returned document.
- The LINK REL attribute MUST indicate that this is of type "dix:/homesite-path".
- The LINK HREF attribute MUST contain the Homsite Path that is authoritative for the Persona.

Here’s an example

```html
<LINK REL="dix:/homesite-path" HREF="http://www.sxip.net/homesite"/>
```
5.10.3.3. Verify Request Message

Sent by a Membersite directly to a Homesite. The message parameters are:

dix:/message-type, required: ‘dix:/verify-request’

dix:/signature, required: The response signature from the response message being verified.

dix:/digest, required: The digest of the response message being verified. Created using the Digest Generation algorithm.

5.10.3.4. Verify Response Message

The body of the response is of content-type text/plain and contains a single DIX URI. The grammar for the body is.

```
verify_response = ( "dix:/true" / "dix:/false" / "dix:/unknown" )
```

The meaning of the response is described in the following table. The response must be followed by a Newline character, and all subsequent characters are to be ignored by the recipient. This could optionally be human readable text to aide the user or developer.

The possible responses are:

dix:/true: Message passed verification check.

dix:/false: Message failed verification check.

dix:/unknown: Message could not be verified. Could be due to communications errors, system errors, service unavailability, etc.

5.10.3.5. Membersite’s Verification Algorithm

1. Receive fetch-response message from Homesite.

2. Check that the datetime provide in ‘dix:/message-datetime’ is within the message age policy.

3. Check the nonce provided in ‘dix:/message-id’ is correct.
4. Check that 'dix:/homesite-url' is authoritative for the 'dix:/persona-url'.

5. Create verify message, using the Digest Generation algorithm to create the value for 'dix:/digest', and the 'dix:/signature' from the fetch-response message.

6. Send verify request message to Homesite.


5.10.3.6. Homesite’s Verification Algorithm

1. Receive verify request message from Membersite.

2. Compute comparison signature based on 'dix:/digest-sha1' provided. Compare with 'dix:/signature'.

3. Create verify response with the body text being 'dix:/true', 'dix:/false', or 'dix:/unknown.

6. Security Considerations

6.1.1. HTTP and HTTPS

Persona URLs and Homesite URLs could use the HTTPS protocol scheme, which would provide more security.

6.1.2. Man in the Middle:

6.1.2.1. Message Replay

The 'dix:/message-id' fetch-request message parameter provided by the Membersite serves as a nonce, which guards against a reply attack, assuming proper implementation by the Membersite.

6.1.2.2. Message Alteration

When a Membersite receives a fetch-response message it resolves the Persona URL provided to get a document that contains a persona delegation tag which provides a set of Homesites that are authoritative for that persona. If the fetch-response message contains a Homesite URL that is not found in that set then the message is not valid. The Membersite then verifies the response message by sending a verify request message to the Homesite containing the digest from the fetch-response message.
If a MITM were to change the Homesite URL then the message would be found to be invalid. If a MITM were to change the Persona URL then the digest would be found invalid by the Homesite. If the MITM were to change both the Persona URL and the Homesite URL then the MITM will be changing the identifier so can’t compromise the original identity.

7. IANA Considerations

This document has no IANA Actions.

8. Acknowledgments

The engineering team at Sxip Identity Corporation.
References

7.1. Normative References


7.2. Informative References

Author’s Addresses

John Merrells
Sxip Identity Corporation
Suite 206 - 55 Water St
Vancouver, BC
Canada V6B 1A1

Email: merrells@sxip.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information
on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2006).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.