```
<h1>Credential Management Level 1</h1>
<pre class="metadata">
Status: ED
ED: https://w3c.github.io/webappsec-credential-management/
TR: http://www.w3.org/TR/credential-management-1/
Previous Version: http://www.w3.org/TR/2015/WD-credential-management-1-20150430/
Shortname: credential-management
Level: 1
Editor: Mike West 56384, Google Inc., mkwst@google.com
Group: webappsec
Abstract:
  This specification describes an imperative API enabling a website to request a
  user's credentials from a user agent, and to help the user agent correctly
  store user credentials for future use.
Indent: 2
Version History: https://github.com/w3c/webappsec-credential-management/commits/master/index.src.html
Boilerplate: omit conformance, omit feedback-header
!Participate: <a href="https://github.com/w3c/webappsec-credential-management/issues/new">File an issue</a> (<a href="https://github.com/w3c/webappsec-credential-management/issues">open issues</a>)
Markup Shorthands: css off, markdown on
</pre>
<pre class="anchors">
spec: ECMA262; urlPrefix: https://tc39.github.io/ecma262/
  type: dfn
    text: JavaScript realm; url: sec-code-realms
spec: HTML; urlPrefix: https://html.spec.whatwg.org/multipage/
  urlPrefix: forms.html
    type: element-attr
      text: autocomplete; for: input; url: #attr-fe-autocomplete
      text: name; for: input; url: #attr-fe-name
      text: enctype; for: form; url: #concept-fs-enctype
    type: attr-value
      for: autocomplete
        text: current-password; url: attr-fe-autocomplete-current-password
        text: new-password; url: attr-fe-autocomplete-new-password
        text: nickname; url: attr-fe-autocomplete-nickname
        text: name; url: attr-fe-autocomplete-name
        text: photo; url: attr-fe-autocomplete-photo
        text: username; url: attr-fe-autocomplete-username
spec: XHR; urlPrefix: https://xhr.spec.whatwg.org/
  type: dfn
    text: entry; url: concept-formdata-entry
    text: entries; for: FormData; url: concept-formdata-entry
    text: name; for: entry; url: concept-formdata-entry-name
    text: value; for: entry; url: concept-formdata-entry-value
    text: type; for: entry; url: concept-formdata-entry-type
  type: interface
    text: FormData; url: interface-formdata
spec: PSL; urlPrefix: https://publicsuffix.org/list/
  type: dfn
    text: registerable domain; url: #
    text: public suffix; url: #
spec: FETCH; urlPrefix: https://fetch.spec.whatwg.org/
  type: dfn
    text: http-network-or-cache fetch; url: http-network-or-cache-fetch
</pre>
<pre class="link-defaults">
spec:html; type:dfn; for:/; text:origin
spec:fetch; type:dfn; for:/; text:request
spec:fetch; type:dictionary; for:/; text:RequestInit
```

```
<!-- These need to be exported -->
spec:html; type:dfn; text:submittable element
spec:html; type:dfn; text:form owner
```



```
<h1>Credential Management Level 1</h1>
<pre class="metadata">
Status: ED
ED: https://w3c.github.io/webappsec-credential-management/
TR: http://www.w3.org/TR/credential-management-1/
Previous Version: http://www.w3.org/TR/2015/WD-credential-management-1-20150430/
Shortname: credential-management
Level: 1
Editor: Mike West 56384, Google Inc., mkwst@google.com
Group: webappsec
Abstract:
  This specification describes an imperative API enabling a website to request a
  user's credentials from a user agent, and to help the user agent correctly
  store user credentials for future use.
Indent: 2
Version History: https://github.com/w3c/webappsec-credential-management/commits/master/index.src.html
Boilerplate: omit conformance, omit feedback-header
!Participate: <a href="https://github.com/w3c/webappsec-credential-management/issues/new">File an issue</a> (<a href="https://github.com/w3c/webappsec-credential-management/issues">open issues</a>)
Markup Shorthands: css off, markdown on
</pre>
<pre class="anchors">
spec: ECMA262; urlPrefix: https://tc39.github.io/ecma262/
  type: dfn
    text: JavaScript realm; url: sec-code-realms
spec: HTML; urlPrefix: https://html.spec.whatwg.org/multipage/
  urlPrefix: forms.html
    type: element-attr
      text: autocomplete; for: input; url: #attr-fe-autocomplete
      text: name; for: input; url: #attr-fe-name
      text: enctype; for: form; url: #concept-fs-enctype
    type: attr-value
      for: autocomplete
        text: current-password; url: attr-fe-autocomplete-current-password
        text: new-password; url: attr-fe-autocomplete-new-password
        text: nickname; url: attr-fe-autocomplete-nickname
        text: name; url: attr-fe-autocomplete-name
        text: photo; url: attr-fe-autocomplete-photo
        text: username; url: attr-fe-autocomplete-username
spec: XHR; urlPrefix: https://xhr.spec.whatwg.org/
  type: dfn
    text: entry; url: concept-formdata-entry
    text: entries; for: FormData; url: concept-formdata-entry
    text: name; for: entry; url: concept-formdata-entry-name
    text: value; for: entry; url: concept-formdata-entry-value
    text: type; for: entry; url: concept-formdata-entry-type
  type: interface
    text: FormData; url: interface-formdata
spec: PSL; urlPrefix: https://publicsuffix.org/list/
  type: dfn
    text: registerable domain; url: #
    text: public suffix; url: #
spec: FETCH; urlPrefix: https://fetch.spec.whatwg.org/
  type: dfn
    text: http-network-or-cache fetch; url: http-network-or-cache-fetch
</pre>
<pre class="link-defaults">
spec:html; type:dfn; for:/; text:origin
spec:fetch; type:dfn; for:/; text:request
spec:fetch; type:dictionary; for:/; text:RequestInit
spec:infra; type:dfn; for:/; text:set
spec:infra; type:dfn; for:struct; text:item
spec:webidl; type:idl; for:/; text:Function

<!-- These need to be exported -->
spec:html; type:dfn; text:submittable element
spec:html; type:dfn; text:form owner
```

```
spec:html; type:dfn; text:autofill detail tokens
spec:url; type:dfn; text:urlencoded byte serializer
</pre>



<pre class="biblio">
{
  "WEB-LOGIN": {
    "authors": [ "Jason Denizac", "Robin Berjon", "Anne van Kesteren" ],
    "href": "https://github.com/jden/web-login",
    "title": "web-login"
  }
}
</pre>

<!--
[redacted]
-->
<section>
  # Introduction # {#introduction}

  <em>This section is not normative.</em>

  Signing into websites is more difficult than it should be. The user agent is in a unique position
  to improve the experience in a number of ways, and most modern user agents have recognized this by
  providing some measure of  credential management natively in the browser. Users can save usernames
  and passwords for websites, and those [=credentials=] are autofilled into sign-in forms with
  varying degrees of success.

  The <{input/autocomplete}> attribute offers a declarative mechanism by which websites can work
  with user agents to improve the latter's ability to detect and fill sign-in forms by marking
  specific fields as "username" or "password", and user agents implement a wide variety of detection
  heuristics to work with websites which haven't taken the time to provide this detail in markup.

  While this combination of heuristic and declarative detection works relatively well, the status
  quo leaves some large gaps where detection is problematic. Sites with uncommon sign-in
  mechanisms (submitting credentials via {{XMLHttpRequest}} [[XMLHTTPREQUEST]], for instance) are
  difficult to reliably detect, as is the increasingly common case in which users wish to
  authenticate themselves using a federated identity provider. Allowing websites to more directly
  interact with the user agent's credential manager would allow the credential manager to be more
  accurate on the one hand, and to assist users with federated sign-in on the other.

  These use cases are explored in more detail in [[#use-cases]] and in
  <a href="https://w3c.github.io/webappsec/usecases/credentialmanagement/">Credential Management:
  Use Cases and Requirements</a>; this specification attempts to address many of the requirements
  that document outlines by defining a Credential Manager API which a website can use to request
  [=credentials=] for a user, and to ask the user agent to persist credentials when a user signs in
  successfully.

  Note: The API defined here is intentionally small and simple: it does not intend to provide
  authentication in and of itself, but is limited to providing an interface to the existing
  credential managers implemented by existing user agents. That functionality is valuable
  <em>right now</em>, without significant effort on the part of either vendors or authors. There's
  certainly quite a bit more which could be done, of course. See [[#teh-futur]] for some thoughts
  we've punted for now, but which could be explored in future iterations of this API.

  ## Use Cases ## {#use-cases}

  Modern user agents generally offer users the capability to save passwords when signing into a
  website, and likewise offer the capability to fill those passwords into sign-in forms fully- or
  semi-automatically when users return to a website. From the perspective of a website, this
```

```
spec:html; type:dfn; text:autofill detail tokens
spec:url; type:dfn; text:urlencoded byte serializer
</pre>
<pre class='ignored-specs'>
spec:css-syntax-3;
</pre>
<pre class="biblio">
{
  "WEB-LOGIN": {
    "authors": [ "Jason Denizac", "Robin Berjon", "Anne van Kesteren" ],
    "href": "https://github.com/jden/web-login",
    "title": "web-login"
  }
}
</pre>

<!--
[redacted]
-->
<section>
  # Introduction # {#introduction}

  <em>This section is not normative.</em>

  Signing into websites is more difficult than it should be. The user agent is in a unique position
  to improve the experience in a number of ways, and most modern user agents have recognized this by
  providing some measure of  credential management natively in the browser. Users can save usernames
  and passwords for websites, and those [=credentials=] are autofilled into sign-in forms with
  varying degrees of success.

  The <{input/autocomplete}> attribute offers a declarative mechanism by which websites can work
  with user agents to improve the latter's ability to detect and fill sign-in forms by marking
  specific fields as "username" or "password", and user agents implement a wide variety of detection
  heuristics to work with websites which haven't taken the time to provide this detail in markup.

  While this combination of heuristic and declarative detection works relatively well, the status
  quo leaves some large gaps where detection is problematic. Sites with uncommon sign-in
  mechanisms (submitting credentials via {{XMLHttpRequest}} [[XMLHTTPREQUEST]], for instance) are
  difficult to reliably detect, as is the increasingly common case in which users wish to
  authenticate themselves using a federated identity provider. Allowing websites to more directly
  interact with the user agent's credential manager would allow the credential manager to be more
  accurate on the one hand, and to assist users with federated sign-in on the other.

  These use cases are explored in more detail in [[#use-cases]] and in
  <a href="https://w3c.github.io/webappsec/usecases/credentialmanagement/">Credential Management:
  Use Cases and Requirements</a>; this specification attempts to address many of the requirements
  that document outlines by defining a Credential Manager API which a website can use to request
  [=credentials=] for a user, and to ask the user agent to persist credentials when a user signs in
  successfully.

  Note: The API defined here is intentionally small and simple: it does not intend to provide
  authentication in and of itself, but is limited to providing an interface to the existing
  credential managers implemented by existing user agents. That functionality is valuable
  <em>right now</em>, without significant effort on the part of either vendors or authors. There's
  certainly quite a bit more which could be done, of course. See [[#teh-futur]] for some thoughts
  we've punted for now, but which could be explored in future iterations of this API.

  ## Use Cases ## {#use-cases}

  Modern user agents generally offer users the capability to save passwords when signing into a
  website, and likewise offer the capability to fill those passwords into sign-in forms fully- or
  semi-automatically when users return to a website. From the perspective of a website, this
```

behavior is completely invisible: the website doesn't know that passwords have been stored, and it isn't notified that passwords have been filled. This is both good and bad. On the one hand, a user agent's password manager works regardless of whether or not a site cooperates, which is excellent for users. On the other, the password managers' behaviors are a fragile and proprietary hodgepodge of heuristics meant to detect and fill sign-in forms, password change forms, etc.

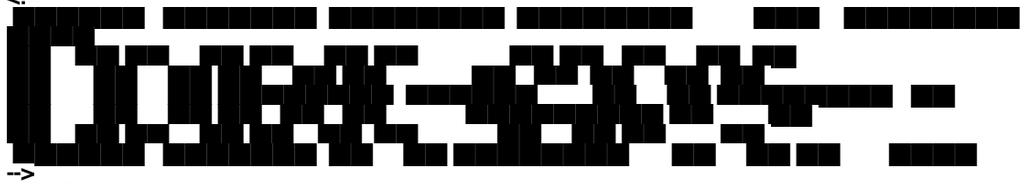
A few problems with the status quo stand out as being particularly noteworthy:

* User agents have an incredibly difficult time helping users with federated identity providers. While detecting a username/password form submission is fairly straightforward, detecting sign-in via a third-party is quite difficult to reliably do well. It would be nice if a website could help the user agent understand the intent behind the redirects associated with a typical federated sign-in action.

* Likewise, user agents struggle to detect more esoteric sign-in mechanisms than simple username/password forms. Authors increasingly asynchronously sign users in via {{XMLHttpRequest}} or similar mechanisms in order to improve the experience and take more control over the presentation. This is good for users, but tough for user agents to integrate into their password managers. It would be nice if a website could help the user agent make sense of the sign-in mechanism they choose to use.

* Finally, changing passwords is less well-supported than it could be if the website explicitly informed the user agent that credentials had changed.

</section>

<!--

-->

<section>
Core API # {#core}

From a developer's perspective, a <dfn export id="concept-credential">credential</dfn> is an object which allows a developer to make an authentication decision for a particular action. This section defines a generic and extensible {{Credential}} interface which serves as a base class for [=credentials=] defined in this and other documents, along with a set of APIs hanging off of {{CredentialsContainer|navigator.credentials.*}} which enable developers to obtain them.

Various types of [=credentials=] are represented to JavaScript as an interface which [=interface/inherits=], either directly or indirectly, from the {{Credential}} interface. This document defines two such interfaces, {{PasswordCredential}} and {{FederatedCredential}}.

A [=credential=] is <dfn export for="credential">effective</dfn> for a particular [=origin=] if it is accepted as authentication on that origin. Even if a credential is effective at a particular point in time, the UA can't assume that the same credential will be effective at any future time, for a couple reasons:

1. A password credential may stop being effective if the account holder changes their password.
2. A credential made from a token received over SMS is likely to only be effective for a single use.

Single-use [=credentials=] are generated by a <dfn export>credential source</dfn>, which could be a private key, access to a federated account, the ability to receive SMS messages at a particular phone number, or something else. Credential sources are not exposed to Javascript or explicitly represented in this specification. To unify the model, we consider a password to be a credential source on its own, which is simply copied to create password credentials.

Even though the UA can't assume that an effective credential will still be effective if used a

```
second time, or that a credential source that has generated an effective credential will be able
to generate a second effective credential in the future, the second is more likely than the first.
By recording (with {{CredentialsContainer/store()}}) which credentials have been effective in the
past, the UA has a better chance of [offering](#user-mediated-selection) effective credential
sources to the user in the future.

## Infrastructure ## {#core-infrastructure}

User agents MUST internally provide a <dfn export id="concept-credential-store">credential
store</dfn>, which is a vendor-specific, opaque storage mechanism to record which [=credentials=]
have been [=effective=]. It offers the following capabilities for [=credential=] access and
persistence:

1.  <dfn for="credential store" abstract-op export>Store a credential</dfn> for later retrieval.
    This accepts a [=credential=], and inserts it into the [=credential store=].

2.  <dfn for="credential store" abstract-op export>Retrieve a list of credentials</dfn>. This
    accepts an arbitrary filter, and returns a set of [=credentials=] that match the filter.

3.  <dfn for="credential store" abstract-op export>Modify a credential</dfn>. This accepts a
    [=credential=], and overwrites the state of an existing [=credential=] in the
    [=credential store=].

Additionally, the [=credential store=] should maintain a <dfn for="origin">`prevent silent
access` flag</dfn> for [=origins=] (which is set to `true` unless otherwise specified).
An origin <dfn for="origin">requires user mediation</dfn> if its flag is set to `true`.

Note: The importance of user mediation is discussed in more detail in [[#user-mediation]].

Note: The [=credential store=] is an internal implementation detail of a user agent's
implementation of the API specified in this document, and is not exposed to the web directly.
More capabilities may be specified by other documents in support of specific [=credential=]
types.

This document depends on the Infra Standard for a number of foundational concepts used in its
algorithms and prose [[!INFRA]].

## The `Credential` Interface ## {#the-credential-interface}

<pre class="idl">
  [Exposed=Window, SecureContext]
  interface Credential {
    readonly attribute USVString id;
    readonly attribute DOMString type;
  };
</pre>
<dl dfn-for="Credential">
  :  <dfn attribute>id</dfn>
  ::  The credential's identifier. The requirements for the identifier are distinct for each type
      of [=credential=]. It might represent a username for username/password tuples, for example.

  :  <dfn attribute>type</dfn>
  ::  This attribute's getter returns the value of the object's [=interface object=]'s
      {{[[type]]}} slot, which specifies the kind of credential represented by this object.

  :  <dfn attribute>\[[type]]</dfn>
  ::  The {{Credential}} [=interface object=] has an internal slot named `[[type]]`, which
      unsurprisingly contains a string representing the type of the credential. The slot's value
      is the empty string unless otherwise specified.

      Note: The {{[[type]]}} slot's value will be the same for all credentials implementing a
      particular interface, which means that developers can rely on `obj.type` returning a string
      that unambigiously represents the specific kind of {{Credential}} they're dealing with.

  :  <dfn attribute>\[[discovery]]</dfn>
  ::  The {{Credential}} [=interface object=] has an internal slot named `[[discovery]]`,
      representing the mechanism by which the user agent can collect credentials of
      a given type. Its value is either
      "<dfn enum-value for="Credential/[[discovery]]">`credential store`</dfn>" or
```

```
        "<dfn enum-value for="Credential/[[discovery]]">`remote`</dfn>". The former value means that
        all available credential information is stored in the user agent's [=credential store=],
        while the latter means that the user agent can discover credentials outside of those
        explicitly represented in the [=credential store=] via interaction with some external device
        or service.
    </dl>

    ISSUE: Talk to Tobie/Dominic about the [=interface object=] bits, here and in
    [[#algorithm-request]], etc. I'm not sure I've gotten the terminology right. [=interface prototype
    object=], maybe?

    Some {{Credential}} objects are <dfn for="Credential">origin bound</dfn>: these contain an
    internal slot named <dfn for="Credential" attribute>\[[origin]]</dfn>, which stores the [=origin=]
    for which the {{Credential}} may be [=effective=].

    ### `Credential` Internal Methods ### {#credential-internal-methods}

    Each [=interface object=] created for interfaces which [=interface/inherit=] from {{Credential}}
    defines several internal methods that allow retrieval and storage of {{Credential}} objects:

    <section algorithm="'collect credentials' internal method">

      <dfn for="Credential" method export>\[[CollectFromCredentialStore]](origin, options)</dfn> is called
      with an [=environment settings object/origin=] and a {{CredentialRequestOptions}}, and returns
      a set of {{Credential}} objects from the user
      agent's [=credential store=] that match the options provided. If no matching {{Credential}}
      objects are available, the returned set will be empty.

      <ul class="algorithm">
        1.  Return an empty set.
      </ul>
    </section>

    <section algorithm="'discover credentials' internal method">

      <dfn for="Credential" method export>\[[DiscoverFromExternalSource]](origin, options)</dfn> is
    called
      with an [=environment settings object/origin=] and a {{CredentialRequestOptions}} object.
      It returns a {{Credential}} if one can be
      returned given the options provided, `null` if no credential is available, or an error if
      discovery fails (for example, incorrect options could produce a {{TypeError}}). If this
      kind of {{Credential}} is only [=effective=] for a single use or a limited time, this
      method is responsible for generating new [=credentials=] using a [=credential source=].

      <ul class="algorithm">
        1.  Return `null`.
      </ul>
    </section>

    <section algorithm="'store a credential' internal method">
      <dfn for="Credential" method export>\[[Store]](credential)</dfn> is called with a {{Credential}},
      and returns once {{Credential}} is persisted to the [=credential store=]:

      <ul class="algorithm">
        1.  Return.
```

```
        "<dfn enum-value for="Credential/[[discovery]]">`remote`</dfn>". The former value means that
        all available credential information is stored in the user agent's [=credential store=],
        while the latter means that the user agent can discover credentials outside of those
        explicitly represented in the [=credential store=] via interaction with some external device
        or service.
    </dl>

    ISSUE: Talk to Tobie/Dominic about the [=interface object=] bits, here and in
    [[#algorithm-request]], etc. I'm not sure I've gotten the terminology right. [=interface prototype
    object=], maybe?

    Some {{Credential}} objects are <dfn for="Credential">origin bound</dfn>: these contain an
    internal slot named <dfn for="Credential" attribute>\[[origin]]</dfn>, which stores the [=origin=]
    for which the {{Credential}} may be [=effective=].

    ### `Credential` Internal Methods ### {#credential-internal-methods}

    The {{Credential}} [=interface object=] features several internal methods facilitating
    retrieval and storage of {{Credential}} objects, with default "no-op" implementations
    as specified in this section, below.

    Unless otherwise specified, each [=interface object=] created for interfaces which [=interface/inherit=]
    from {{Credential}} MUST provide implementations for at least one of these internal methods,
    overriding
    {{Credential}}'s default implementations, as appropriate for the [=credential=] type. E.g.,
    [[#passwordcredential-interface]] and [[#federatedcredential-interface]].

    <h5 id="algorithm-collect-creds" algorithm>`[[CollectFromCredentialStore]]` internal method</h5>
      <dfn for="Credential" method export>\[[CollectFromCredentialStore]](origin, options)</dfn> is called
      with an [=environment settings object/origin=] and a {{CredentialRequestOptions}}, and returns
      a set of {{Credential}} objects from the user
      agent's [=credential store=] that match the options provided. If no matching {{Credential}}
      objects are available, the returned set will be empty.

      {{Credential}}'s default implementation of {{Credential/[[CollectFromCredentialStore]](origin,
    options)}}:

      <ul class="algorithm">
        1.  Return an empty set.
      </ul>

    <h5 id="algorithm-discover-creds" algorithm>`[[DiscoverFromExternalSource]]` internal method</h5>
      <dfn for="Credential" method export>\[[DiscoverFromExternalSource]](origin, options)</dfn> is
    called
      with an [=environment settings object/origin=] and a {{CredentialRequestOptions}} object.
      It returns a {{Credential}} if one can be
      returned given the options provided, `null` if no credential is available, or an error if
      discovery fails (for example, incorrect options could produce a {{TypeError}}). If this
      kind of {{Credential}} is only [=effective=] for a single use or a limited time, this
      method is responsible for generating new [=credentials=] using a [=credential source=].

      {{Credential}}'s default implementation of {{Credential/[[DiscoverFromExternalSource]](origin,
    options)}}:

      <ul class="algorithm">
        1.  Return `null`.
      </ul>

    <h5 id="algorithm-store-cred" algorithm>`[[Store]]` internal method</h5>
      <dfn for="Credential" method export>\[[Store]](credential)</dfn> is called with a {{Credential}},
      and returns once {{Credential}} is persisted to the [=credential store=].

      {{Credential}}'s default implementation of {{Credential/[[Store]](credential)}}:

      <ul class="algorithm">
        1.  Return.
```

```
    </ul>
  </section>

  <section algorithm="'create a credential' internal method">
    <dfn for="Credential" method export>\[[Create]](options)</dfn> is called with a
    {{CredentialCreationOptions}}, and returns either a {{Credential}}, if one can be created,
    `null` if no credential was created, or an error if creation fails due to exceptional situations
    (for example, incorrect options could produce a {{TypeError}}):

    <ul class="algorithm">
      1. Return `null`.
    </ul>
  </section>

  Unless otherwise specified, the [=interface objects=] for [=interfaces=] which [=interface/inherit=] from
  {{Credential}} will alias {{Credential}}'s implementation of these three internal methods. Since that
  implementation isn't particularly useful, derived interfaces MUST override one or the other.

  ### `CredentialUserData` Mixin ### {#credentialuserdata-mixin}

  Some {{Credential}} objects contain data which aims to give users a human-readable disambiguation
  mechanism in the [=credential chooser=] by providing a friendly name and icon:

  <pre class="idl">
    [NoInterfaceObject, SecureContext]
    interface CredentialUserData {
      readonly attribute USVString name;
      readonly attribute USVString iconURL;
    };
  </pre>
  <dl dfn-for="CredentialUserData">
    :  <dfn attribute>name</dfn>
    ::  A name associated with the credential, intended as a human-understandable public name for
        display in a [=credential chooser=].
    :  <dfn attribute>iconURL</dfn>
    ::  A URL pointing to an image for the credential, intended for display in a
        [=credential chooser=]. This URL MUST be an <a><i lang="la">a priori</i> authenticated
        URL</a>.
  </dl>

  ## `navigator.credentials` ## {#framework-credential-management}

  Developers retrieve {{Credential}}s and interact with the user agent's [=credential store=] via
  methods exposed on the
  {{CredentialsContainer}} interface, which hangs off the {{Navigator}} object as
  <a attribute for="Navigator" lt="credentials">`navigator.credentials `</a>.
```



```
    </ul>

  <h5 id="algorithm-create-cred" algorithm>`[[Create]]` internal method</h5>
    <dfn for="Credential" method export>\[[Create]](origin, options)</dfn> is called
    [=in parallel=] with an [=environment settings object/origin=] and a {{CredentialCreationOptions}},
    and either:

    * creates a {{Credential}}, or,
    * does not create a credential and returns `null`, or,
    * returns an error if creation fails due to exceptional situations
      (for example, incorrect options could produce a {{TypeError}}).

    When creating a {{Credential}}, it will either:

    * directly return a {{Credential}} (in which case it accomplished everything while
      running [=in parallel=]), or,
    * return a [=constructCredential=] algorithm which yields a {{Credential}} when
      subsequently invoked from a [=task=].

    {{Credential}}'s default implementation of {{Credential/[[Create]](origin, options)}}:

    <ul class="algorithm">
      1. Return `null`.
    </ul>

  <h6 id="algorithm-construct-cred">constructCredential Algorithm</h6>

    A <dfn>constructCredential</dfn> algorithm is a [=queue=] of steps
      to be invoked by a [=task=] from the [=DOM manipulation task source=].
      [=constructCredential=]'s purpose is to construct an [=interface object=] derived
      from {{Credential}}, given a [=current settings object=]'s
      [=environment settings object/global object=].

    Note: <code>[=constructCredential=]</code>'s algorithm steps are defined on a
      per-[=credential=] type basis by those [=credential=] type specifications needing to
      employ it. E.g., [[WEBAUTHN]].

  ### `CredentialUserData` Mixin ### {#credentialuserdata-mixin}

  Some {{Credential}} objects contain data which aims to give users a human-readable disambiguation
  mechanism in the [=credential chooser=] by providing a friendly name and icon:

  <pre class="idl">
    [NoInterfaceObject, SecureContext]
    interface CredentialUserData {
      readonly attribute USVString name;
      readonly attribute USVString iconURL;
    };
  </pre>
  <dl dfn-for="CredentialUserData">
    :  <dfn attribute>name</dfn>
    ::  A name associated with the credential, intended as a human-understandable public name for
        display in a [=credential chooser=].
    :  <dfn attribute>iconURL</dfn>
    ::  A URL pointing to an image for the credential, intended for display in a
        [=credential chooser=]. This URL MUST be an <a><i lang="la">a priori</i> authenticated
        URL</a>.
  </dl>

  ## `navigator.credentials` ## {#framework-credential-management}

  Developers retrieve {{Credential}}s and interact with the user agent's [=credential store=] via
  methods exposed on the
  {{CredentialsContainer}} interface, which hangs off the {{Navigator}} object as
  <a attribute for="Navigator" lt="credentials">`navigator.credentials `</a>.
```

```
<pre class="idl">
  partial interface Navigator {
    [SecureContext, SameObject] readonly attribute CredentialsContainer credentials;
  };
</pre>

The <dfn for="Navigator" attribute>`credentials`</dfn> attribute MUST return the
{{CredentialsContainer}} associated with the [=context object=].

Note: As discussed in [[#insecure-sites]], the credential management API is exposed only in
[=Secure Contexts=].

<pre class="idl">
  [Exposed=Window, SecureContext]
  interface CredentialsContainer {
    Promise&lt;Credential?&gt; get(optional CredentialRequestOptions options);
    Promise&lt;Credential&gt; store(Credential credential);
    Promise&lt;Credential?&gt; create(optional CredentialCreationOptions options);
    Promise&lt;void&gt; preventSilentAccess();
  };

  dictionary CredentialData {
    required USVString id;
  };
</pre>

<dl dfn-for="CredentialsContainer">
  :  <dfn method>get(options)</dfn>
  ::  When {{CredentialsContainer/get()}} is called, the user agent MUST return the result of
      executing <a abstract-op>Request a `Credential`</a> on
      {{CredentialsContainer/get(options)/options}}.

      <pre class="argumentdef" for="CredentialsContainer/get(options)">
        options: The set of properties governing the scope of the request.
      </pre>

  :  <dfn method lt="store(credential)|store()">store(credential)</dfn>
  ::  When {{CredentialsContainer/store()}} is called, the user agent MUST return the result of
      executing <a abstract-op>Store a `Credential`</a> on
      {{CredentialsContainer/store(credential)/credential}}.

      <pre class="argumentdef" for="CredentialsContainer/store(credential)">
        credential: The credential to be stored.
      </pre>

  :  <dfn method lt="create(options)|create()">create(options)</dfn>
  ::  When {{CredentialsContainer/create()}} is called, the user agent MUST return the result of
      executing <a abstract-op>Create a `Credential`</a> on
      {{CredentialsContainer/create(options)/options}}.

      <pre class="argumentdef" for="CredentialsContainer/create(options)">
        options: The options used to create a `Credential`.
      </pre>

  :  <dfn method>preventSilentAccess()</dfn>
  ::  When {{CredentialsContainer/preventSilentAccess()}} is called, the user agent MUST return
      the result of executing <a abstract-op>Prevent Silent Access</a> on the <a>current settings
      object</a>.

      Note: The intent here is a signal from the origin that the user has signed out. That
      is, after a click on a "Sign out" button, the site updates the user's session info, and
      calls `navigator.credentials.preventSilentAccess()`. This sets the <a>`prevent silent
      access` flag</a>, meaning that credentials will not be automagically handed back to the
      page next time the user visits.

      Note: This function was previously called `requireUserMediation()` which should be considered
      deprecated.
</dl>
```



```
<pre class="idl">
  partial interface Navigator {
    [SecureContext, SameObject] readonly attribute CredentialsContainer credentials;
  };
</pre>

The <dfn for="Navigator" attribute>`credentials`</dfn> attribute MUST return the
{{CredentialsContainer}} associated with the [=context object=].

Note: As discussed in [[#insecure-sites]], the credential management API is exposed only in
[=Secure Contexts=].

<pre class="idl">
  [Exposed=Window, SecureContext]
  interface CredentialsContainer {
    Promise&lt;Credential?&gt; get(optional CredentialRequestOptions options);
    Promise&lt;Credential&gt; store(Credential credential);
    Promise&lt;Credential?&gt; create(optional CredentialCreationOptions options);
    Promise&lt;void&gt; preventSilentAccess();
  };

  dictionary CredentialData {
    required USVString id;
  };
</pre>

<dl dfn-for="CredentialsContainer">
  :  <dfn method>get(options)</dfn>
  ::  When {{CredentialsContainer/get()}} is called, the user agent MUST return the result of
      executing <a abstract-op>Request a `Credential`</a> on
      {{CredentialsContainer/get(options)/options}}.

      <pre class="argumentdef" for="CredentialsContainer/get(options)">
        options: The set of properties governing the scope of the request.
      </pre>

  :  <dfn method lt="store(credential)|store()">store(credential)</dfn>
  ::  When {{CredentialsContainer/store()}} is called, the user agent MUST return the result of
      executing <a abstract-op>Store a `Credential`</a> on
      {{CredentialsContainer/store(credential)/credential}}.

      <pre class="argumentdef" for="CredentialsContainer/store(credential)">
        credential: The credential to be stored.
      </pre>

  :  <dfn method lt="create(options)|create()">create(options)</dfn>
  ::  When {{CredentialsContainer/create()}} is called, the user agent MUST return the result of
      executing <a abstract-op>Create a `Credential`</a> on
      {{CredentialsContainer/create(options)/options}}.

      <pre class="argumentdef" for="CredentialsContainer/create(options)">
        options: The options used to create a `Credential`.
      </pre>

  :  <dfn method>preventSilentAccess()</dfn>
  ::  When {{CredentialsContainer/preventSilentAccess()}} is called, the user agent MUST return
      the result of executing <a abstract-op>Prevent Silent Access</a> on the <a>current settings
      object</a>.

      Note: The intent here is a signal from the origin that the user has signed out. That
      is, after a click on a "Sign out" button, the site updates the user's session info, and
      calls `navigator.credentials.preventSilentAccess()`. This sets the <a>`prevent silent
      access` flag</a>, meaning that credentials will not be automagically handed back to the
      page next time the user visits.

      Note: This function was previously called `requireUserMediation()` which should be considered
      deprecated.
</dl>
```

```
<div algorithm="create credentialscontainer">
  When a {{Navigator}} object |navigator| is created, the user agent MUST create a
  new {{CredentialsContainer}} object, using |navigator|'s [=relevant Realm=], and associate it
  with |navigator|.
</div>

### The `CredentialRequestOptions` Dictionary ### {#credentialrequestoptions-dictionary}

In order to retrieve a {{Credential}} via {{CredentialsContainer/get()}}, the caller specifies a
few parameters in a {{CredentialRequestOptions}} object.

Note: The {{CredentialRequestOptions}} dictionary is an extension point. If and when new types of
credentials are introduced that require options, their dictionary types will be added to the
dictionary so they can be passed into the request. See [[#implementation-extension]].

<pre class="idl">
  dictionary CredentialRequestOptions {
    CredentialMediationRequirement mediation = "optional";
  };
</pre>
<dl dfn-for="CredentialRequestOptions" dfn-type="dict-member">
  :  <dfn>mediation</dfn>
  ::  This property specifies the mediation requirements for a given credential request. The
      meaning of each enum value is described below in {{CredentialMediationRequirement}}.
      Processing details are defined in [[#algorithm-request]].
</dl>

<div class="note">
  Earlier versions of this specification defined a boolean `unmediated` member. Setting that
  to `true` had the effect of setting {{CredentialRequestOptions/mediation}} to
  "{{CredentialMediationRequirement/silent}}", and setting it to `false` had the effect of
  setting {{CredentialRequestOptions/mediation}} to "{{CredentialMediationRequirement/optional}}".

  `unmediated` should be considered deprecated; new code should instead rely on
  {{CredentialRequestOptions/mediation}}.
</div>

<div algorithm="relevant credential interfaces">
  The <dfn for="CredentialRequestOptions">relevant credential interface objects</dfn> for a given
  {{CredentialRequestOptions}} |options| is a set of [=interface objects=], collected as follows:

  <ol>
    1.  Let |settings| be the <a>current settings object</a>

    2.  Let |interface objects| be the set of [=interface objects=] on |settings|'
        [=environment settings object/global object=].

    3.  Let |relevant interface objects| be an empty set.

    4.  For each |object| in |interface objects|:

        1.  If |object|'s [=inherited interfaces=] do not <a for="set">contain</a> {{Credential}},
            <a for="iteration">continue</a>.

        2.  Let |key| be |object|'s {{Credential/[[type]]}} slot's value.

        3.  If |options|[|key|] <a for="map">exists</a>, <a for="set">append</a> |object| to
            |relevant interface objects|.
  </ol>

  ISSUE: jyasskin@ suggests replacing the iteration through the interface objects with a registry.
  I'm not sure which is less clear, honestly. I'll leave it like this for the moment, and we can
  argue about whether this is too much of a `COMEFROM` interface.
</div>

<div algorithm="can collect locally">
  A given {{CredentialRequestOptions}} |options| is
  <dfn for="CredentialRequestOptions">matchable <i lang="la">a priori</i></dfn> if the following
```

```
    steps return `true`:

    <ol>
      1.  For each |interface| in |options|' [=relevant credential interface objects=]:

          1.  If |interface|'s {{Credential/[[discovery]]}} slot's value is not
              "{{Credential/[[discovery]]/credential store}}", return `false`.

      2.  Return `true`.
    </ol>

    Note: When executing {{get(options)}}, we only return credentials without [=user mediation=] if
    the provided {{CredentialRequestOptions}} is <a>matchable <i lang="la">a priori</i></a>. If any
    credential types are requested that could require discovery from some external service (OAuth
    tokens, security key authenticators, etc.), then [=user mediation=] will be required in order
    to guide the discovery process (by choosing a federated identity provider, BTLE device, etc).
  </div>

  ### Mediation Requirements ### {#mediation-requirements}

  When making a request via {{get(options)}}, developers can set a case-by-case requirement for
  [=user mediation=] by choosing the appropriate {{CredentialMediationRequirement}} enum value.

  Note: The [[#user-mediation]] section gives more detail on the concept in general, and its
  implications on how the user agent deals with individual requests for a given origin).

  <pre class="idl">
    enum CredentialMediationRequirement {
      "silent",
      "optional",
      "required"
    };
  </pre>
  <dl dfn-for="CredentialMediationRequirement" dfn-type="enum-value">
    :  <dfn>silent</dfn>
    ::  User mediation is suppressed for the given operation. If the operation can be performed
        without user involvement, wonderful. If user involvement is necessary, then the operation
        will return `null` rather than involving the user.

        Note: The intended usage is to support ["Keep me signed-into this site"
        scenarios](#example-mediation-silent), where a developer may wish to silently obtain
        credentials if a user should be automatically signed in, but to delay bothering the user
        with a sign-in prompt until they actively choose to sign-in.

    :  <dfn>optional</dfn>
    ::  If credentials can be handed over for a given operation without user mediation, they will
        be. If [=requires user mediation|user mediation is required=], then the user agent will
        involve the user in the decision.

        Note: This is the default behavior for {{get()}}, and is intended to serve a case where a
        developer has reasonable confidence that a user expects to start a sign-in operation. If
        a user has just clicked "sign-in" for example, then they won't be surprised or confused to
        see a [=credential chooser=] if necessary.

    :  <dfn>required</dfn>
    ::  The user agent will not hand over credentials without [=user mediation=], even if the
        [=prevent silent access flag=] is unset for an origin.

        Note: This requirement is intended to support [reauthentication](#example-mediation-require)
        or [user-switching](#example-mediation-switch) scenarios. Further, the requirement is tied
        to a specific operation, and does not affect the [=prevent silent access flag=] for the
        origin. To set that flag, developers should call {{preventSilentAccess()}}.
  </dl>

  #### Examples #### {#mediation-examples}

  <div class="example" id="example-mediation-silent">
    MegaCorp, Inc. wishes to seamlessly sign in users when possible. They can do so by calling
    {{get()}} for all non-signed in users at some convinient point while a landing page is loading,
```

```
      passing in a {{CredentialRequestOptions/mediation}} member set to
      "{{CredentialMediationRequirement/silent}}". This ensures that users who have opted-into
      dropping the requirements for user mediation (as described in [[#user-mediation-requirement]])
      are signed in, and users who haven't opted-into such behavior won't be bothered with a confusing
      [=credential chooser=] popping up without context:

      <pre>
        window.addEventListener('load', _ =&gt; {
          var c = await navigatior.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer" method lt="get()">get</a>({
            ...,
            <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a for="CredentialMediationRequirement" enum-value>silent</a>'
          });
          if (c) {
            // Hooray! Let's sign the user in using these credentials!
          }
        });
      </pre>
    </div>

    <div class="example" id="example-mediation-optional">
      When a user clicks "Sign In", MegaCorp, Inc. wishes to give them the smoothest possible
      experience. If they have [[#user-mediation-requirement!opted-into]] signing in without
      [=user mediation=], and the user agent can unambigiously choose a credential, great! If
      not, a [=credential chooser=] will be presented.

      <pre>
        document.querySelector('#sign-in').addEventListener('click', e =&gt; {
          var c = await navigatior.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer" method lt="get()">get</a>({
            ...,
            <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a for="CredentialMediationRequirement" enum-value>optional</a>'
          });
          if (c) {
            // Hooray! Let's sign the user in using these credentials!
          }
        });
      </pre>

      Note: MegaCorp, Inc. could also have left off the {{CredentialRequestOptions/mediation}}
      member entirely, as "{{CredentialMediationRequirement/optional}}" is its default.
    </div>

    <div class="example" id="example-mediation-require">
      MegaCorp, Inc. wishes to protect a sensitive operation by requiring a user to reauthenticate
      before taking action. Even if a user has [[#user-mediation-requirement!opted-into]] signing in
      without [=user mediation=], MegaCorp, Inc. can ensure that the user agent requires mediation
      by calling {{get()}} with a {{CredentialRequestOptions/mediation}} member set to
      "{{CredentialMediationRequirement/required}}":

      Note: Depending on the security model of the browser or the credential type, this may require
      the user to authenticate themselves in some way, perhaps by entering a master password, scanning
      a fingerprint, etc. before a credential is handed to the website.

      <pre>
        document.querySelector('#important-form').addEventListener('submit', e =&gt; {
          var c = await navigatior.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer" method lt="get()">get</a>({
            ...,
            <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a for="CredentialMediationRequirement" enum-value>required</a>'
          });
          if (c) {
            // Verify that lcl enables access, and cancel the submission
            // if it doesn't.
          } else {
            e.preventDefault();
```

```
          }
        });
      </pre>
    </div>

    <div class="example" id="example-mediation-switch">
      MegaCorp, Inc. wishes to support signing into multiple user accounts at once. In order to ensure
      that the user gets a chance to select a different credential, MegaCorp, Inc. can call {{get()}}
      with a {{CredentialRequestOptions/mediation}} member set to
      "{{CredentialMediationRequirement/required}}" in order to ensure that that credentials aren't
      returned automatically in response to clicking on an "Add account" button:

      <pre>
        document.querySelector('#switch-button').addEventListener('click', e =&gt; {
          var c = await navigatior.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer"
    method lt="get()">get</a>({
            ...,
            <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a
    for="CredentialMediationRequirement" enum-value>required</a>'
          });
          if (c) {
            // Sign the user in using lcl.
          }
        });
      </pre>

    </div>
    ## The `CredentialCreationOptions` Dictionary ## {#credentialcreationoptions-dictionary}

    In order to create a {{Credential}} via {{CredentialsContainer/create()}}, the caller specifies a
    few parameters in a {{CredentialCreationOptions}} object.

    Note: The {{CredentialCreationOptions}} dictionary is an extension point. If and when new types of
    credentials are introduced, they will add to the dictionary so they can be passed into the
    creation method. See [[#implementation-extension]], and the extensions introduced in this document:
    [[#passwordcredential-interface]] and [[#federatedcredential-interface]].

    <pre class="idl">
      dictionary CredentialCreationOptions {
      };
    </pre>

    ## Algorithms ## {#algorithms}

    <h4 id="algorithm-request" algorithm>Request a `Credential`</h4>

    The <dfn abstract-op>Request a `Credential`</dfn> algorithm accepts a {{CredentialRequestOptions}}
    (loptionsl), and returns a {{Promise}} that resolves with a {{Credential}} if one can be
    unambigiously obtained, or with `null` if not.

    <ol class="algorithm">
      1.  Let lsettingsl be the <a>current settings object</a>

      2.  Assert: lsettingsl is a [=secure context=].

      3.  Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
          are true:

          1.  lsettingsl does not have a [=environment settings object/responsible document=].

          2.  lsettingsl' [=environment settings object/responsible document=] is not the
              [=active document=] in a [=top-level browsing context=].

      4.  Let lpl be [=a new promise=].

      5.  Let loriginl be the [=current settings object=]'s [=environment settings object/origin=].

      6.  Run the following steps [=in parallel=]:
```



```
          }
        });
      </pre>
    </div>

    <div class="example" id="example-mediation-switch">
      MegaCorp, Inc. wishes to support signing into multiple user accounts at once. In order to ensure
      that the user gets a chance to select a different credential, MegaCorp, Inc. can call {{get()}}
      with a {{CredentialRequestOptions/mediation}} member set to
      "{{CredentialMediationRequirement/required}}" in order to ensure that that credentials aren't
      returned automatically in response to clicking on an "Add account" button:

      <pre>
        document.querySelector('#switch-button').addEventListener('click', e =&gt; {
          var c = await navigatior.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer"
    method lt="get()">get</a>({
            ...,
            <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a
    for="CredentialMediationRequirement" enum-value>required</a>'
          });
          if (c) {
            // Sign the user in using lcl.
          }
        });
      </pre>

    </div>
    ## The `CredentialCreationOptions` Dictionary ## {#credentialcreationoptions-dictionary}

    In order to create a {{Credential}} via {{CredentialsContainer/create()}}, the caller specifies a
    few parameters in a {{CredentialCreationOptions}} object.

    Note: The {{CredentialCreationOptions}} dictionary is an extension point. If and when new types of
    credentials are introduced, they will add to the dictionary so they can be passed into the
    creation method. See [[#implementation-extension]], and the extensions introduced in this document:
    [[#passwordcredential-interface]] and [[#federatedcredential-interface]].

    <pre class="idl">
      dictionary CredentialCreationOptions {
      };
    </pre>

    ## Algorithms ## {#algorithms}

    <h4 id="algorithm-request" algorithm>Request a `Credential`</h4>

    The <dfn abstract-op>Request a `Credential`</dfn> algorithm accepts a {{CredentialRequestOptions}}
    (loptionsl), and returns a {{Promise}} that resolves with a {{Credential}} if one can be
    unambigiously obtained, or with `null` if not.

    <ol class="algorithm">
      1.  Let lsettingsl be the <a>current settings object</a>

      2.  Assert: lsettingsl is a [=secure context=].

      3.  Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
          are true:

          1.  lsettingsl does not have a [=environment settings object/responsible document=].

          2.  lsettingsl' [=environment settings object/responsible document=] is not the
              [=active document=] in a [=top-level browsing context=].

      4.  Let lpl be [=a new promise=].

      5.  Let loriginl be the [=current settings object=]'s [=environment settings object/origin=].

      6.  Run the following steps [=in parallel=]:
```

1. Let |credentials| be the result of <a abstract-op lt="collect local">collecting `Credential`s from the credential store, given |origin| and |options|.

2. If |credentials| is an [=exception=], [=reject=] |p| with |credentials|.

3. If all of the following statements are true, resolve |p| with |credentials|[0] and skip the remaining steps:

    1. |credentials|' [=set/size=] is 1

    2. |origin| does not [=origin/requires user mediation|require user mediation=]

    3. |options| is <a>matchable a priori.

    4. |options|.{{CredentialRequestOptions/mediation}} is not "{{CredentialMediationRequirement/required}}".

    ISSUE: This might be the wrong model. It would be nice to support a site that wished to accept either username/passwords or webauthn-style credentials without forcing a chooser for those users who use the former, and who wish to remain signed in.

4. If |options|' {{CredentialRequestOptions/mediation}} is "{{CredentialMediationRequirement/silent}}", [=resolve=] |p| with `null`, and skip the remaining steps.

5. Let |choice| be the result of <a abstract-op lt="ask to choose">asking the user to choose a `Credential`, given |options| and |credentials|.

6. If |choice| is `null` or a {{Credential}}, [=resolve=] |p| with |choice| and skip the remaining steps.

7. Assert: |choice| is an [=interface object=].

8. Let |result| be the result of executing |choice|'s {{[[DiscoverFromExternalSource]](origin, options)}}, given |origin| and |options|.

9. If |result| is a {{Credential}} or `null`, resolve |p| with |result|.

    Otherwise, [=reject=] |p| with |result|.

7. Return |p|.

<h4 id="algorithm-collect-known" algorithm>Collect `Credential`s from the credential store</h4>

Given an [=environment settings object/origin=] (|origin|) and a {{CredentialRequestOptions}} (|options|), the user agent may <dfn abstract-op local-lt="collect local">collect `Credential`s from the credential store</dfn>, returning a set of {{Credential}} objects stored by the user agent locally that match |options|' filter. If no such {{Credential}} objects are known, the returned set will be empty:

<ol class="algorithm">
1. Let |possible matches| be an empty set.

2. For each |interface| in |options|' <a>relevant credential interface objects:

    1. Let |r| be the result of executing |interface|'s {{Credential/[[CollectFromCredentialStore]](origin, options)}} internal method on |origin| and |options|.

    2. If |r| is an [=exception=], return |r|.

    3. Assert: |r| is a list of [=interface objects=].

    4. For each |c| in |r|:

        1. Append |c| to |possible matches|.

3. Return |possible matches|.

1. Let |credentials| be the result of <a abstract-op lt="collect local">collecting `Credential`s from the credential store, given |origin| and |options|.

2. If |credentials| is an [=exception=], [=reject=] |p| with |credentials|.

3. If all of the following statements are true, resolve |p| with |credentials|[0] and skip the remaining steps:

    1. |credentials|' [=set/size=] is 1

    2. |origin| does not [=origin/requires user mediation|require user mediation=]

    3. |options| is <a>matchable a priori.

    4. |options|.{{CredentialRequestOptions/mediation}} is not "{{CredentialMediationRequirement/required}}".

    ISSUE: This might be the wrong model. It would be nice to support a site that wished to accept either username/passwords or webauthn-style credentials without forcing a chooser for those users who use the former, and who wish to remain signed in.

4. If |options|' {{CredentialRequestOptions/mediation}} is "{{CredentialMediationRequirement/silent}}", [=resolve=] |p| with `null`, and skip the remaining steps.

5. Let |choice| be the result of <a abstract-op lt="ask to choose">asking the user to choose a `Credential`, given |options| and |credentials|.

6. If |choice| is `null` or a {{Credential}}, [=resolve=] |p| with |choice| and skip the remaining steps.

7. Assert: |choice| is an [=interface object=].

8. Let |result| be the result of executing |choice|'s {{[[DiscoverFromExternalSource]](origin, options)}}, given |origin| and |options|.

9. If |result| is a {{Credential}} or `null`, resolve |p| with |result|.

    Otherwise, [=reject=] |p| with |result|.

7. Return |p|.

<h4 id="algorithm-collect-known" algorithm>Collect `Credential`s from the credential store</h4>

Given an [=environment settings object/origin=] (|origin|) and a {{CredentialRequestOptions}} (|options|), the user agent may <dfn abstract-op local-lt="collect local">collect `Credential`s from the credential store</dfn>, returning a set of {{Credential}} objects stored by the user agent locally that match |options|' filter. If no such {{Credential}} objects are known, the returned set will be empty:

<ol class="algorithm">
1. Let |possible matches| be an empty set.

2. For each |interface| in |options|' <a>relevant credential interface objects:

    1. Let |r| be the result of executing |interface|'s {{Credential/[[CollectFromCredentialStore]](origin, options)}} internal method on |origin| and |options|.

    2. If |r| is an [=exception=], return |r|.

    3. Assert: |r| is a list of [=interface objects=].

    4. For each |c| in |r|:

        1. Append |c| to |possible matches|.

3. Return |possible matches|.

```
</ol>

<h4 id="algorithm-store" algorithm>Store a `Credential`</h4>

The <dfn abstract-op>Store a `Credential`</dfn> algorithm accepts a {{Credential}}
(|credential|), and returns a {{Promise}} which resolves once the object is persisted to the
[=credential store=].

<ol class="algorithm">
  1. Let |settings| be the <a>current settings object</a>

  2. Assert: |settings| is a [=secure context=].

  3. Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
     are true:

     1. |settings| does not have a [=environment settings object/responsible document=].

     2. |settings|' [=environment settings object/responsible document=] is not the
        [=active document=] in a [=top-level browsing context=].

  4. Let |p| be [=a new promise=].

  5. Run the following steps [=in parallel=]:

     1. Let |r| be the result of executing |credential|'s [=interface object=]'s
        {{Credential/[[Store]](credential)}} internal method on |credential|.

     2. [=Resolve=] |p| with |r|.

  6. Return |p|.
</ol>

<h4 id="algorithm-create" algorithm>Create a `Credential`</h4>

The <dfn abstract-op>Create a `Credential`</dfn> algorithm accepts a {{CredentialCreationOptions}}
(|options|), and returns a {{Promise}} which resolves with a {{Credential}} if one can be created
using the options provided, or `null` if no {{Credential}} can be created. In exceptional
circumstances, the {{Promise}} may reject with an appropriate exception:

<ol class="algorithm">
  1. Let |settings| be the <a>current settings object</a>

  2. Assert: |settings| is a [=secure context=].

  3. Let |interfaces| be the set of |options|' <a>relevant credential interface</a> objects</a>.

  4. Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
     are true:

     1. |settings| does not have a [=environment settings object/responsible document=].

     2. |settings|' [=environment settings object/responsible document=] is not the
        [=active document=] in a [=top-level browsing context=].

     3. |interfaces|' [=list/size=] is greater than 1.

        Note: It may be reasonable at some point in the future to loosen this restriction, and
        allow the user agent to help the user choose among one of many potential credential
        types in order to support a "sign-up" use case. For the moment, though, we're punting
        on that by restricting the dictionary to a single entry.

  5. Let |p| be [=a new promise=].

  6. Run the following steps [=in parallel=]:
```



```
</ol>

<h4 id="algorithm-store" algorithm>Store a `Credential`</h4>

The <dfn abstract-op>Store a `Credential`</dfn> algorithm accepts a {{Credential}}
(|credential|), and returns a {{Promise}} which resolves once the object is persisted to the
[=credential store=].

<ol class="algorithm">
  1. Let |settings| be the <a>current settings object</a>

  2. Assert: |settings| is a [=secure context=].

  3. Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
     are true:

     1. |settings| does not have a [=environment settings object/responsible document=].

     2. |settings|' [=environment settings object/responsible document=] is not the
        [=active document=] in a [=top-level browsing context=].

  4. Let |p| be [=a new promise=].

  5. Run the following steps [=in parallel=]:

     1. Let |r| be the result of executing |credential|'s [=interface object=]'s
        {{Credential/[[Store]](credential)}} internal method on |credential|.

     2. [=Resolve=] |p| with |r|.

  6. Return |p|.
</ol>

<h4 id="algorithm-create" algorithm>Create a `Credential`</h4>

The <dfn abstract-op>Create a `Credential`</dfn> algorithm accepts a {{CredentialCreationOptions}}
(|options|), and returns a {{Promise}} which resolves with a {{Credential}} if one can be created
using the options provided, or `null` if no {{Credential}} can be created. In exceptional
circumstances, the {{Promise}} may reject with an appropriate exception:

<ol class="algorithm">
  1. Let |settings| be the <a>current settings object</a>.

  2. Assert: |settings| is a [=secure context=].

  3. Let |global| be |settings|' [=environment settings object/global object=].

  4. Let |interfaces| be the [=set=] of |options|' <a>relevant credential interface objects</a>.

  5. Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
     are true:

     1. |settings| does not have a [=environment settings object/responsible document=].

     2. |settings|' [=environment settings object/responsible document=] is not the
        [=active document=] in a [=top-level browsing context=].

     3. |interfaces|' [=list/size=] is greater than 1.

        Note: It may be reasonable at some point in the future to loosen this restriction, and
        allow the user agent to help the user choose among one of many potential credential
        types in order to support a "sign-up" use case. For the moment, though, we're punting
        on that by restricting the dictionary to a single entry.

  6. Let |p| be [=a new promise=].

  7. Let |origin| be |settings|'s [=environment settings object/origin=].
```

1. Let |rl| be the result of executing |interfaces|[0] {{Credential/[[Create]](options)}} internal method on |options|.

2. If |rl| is an [=exception=], [=reject=] |p| with |rl|.

   Otherwise, [=resolve=] |p| with |rl|.

7. Return |p|.

<h4 id="algorithm-prevent-silent-access" algorithm>Prevent Silent Access</h4>

The <dfn abstract-op>Prevent Silent Access</dfn> algorithm accepts an [=environment settings object=] (|settings|), and returns a {{Promise}} which resolves once the `prevent silent access` flag is persisted to the [=credential store=].

<ol class="algorithm">
1. Let |origin| be |settings|' [=environment settings object/origin=].

2. Let |p| be [=a new promise=]

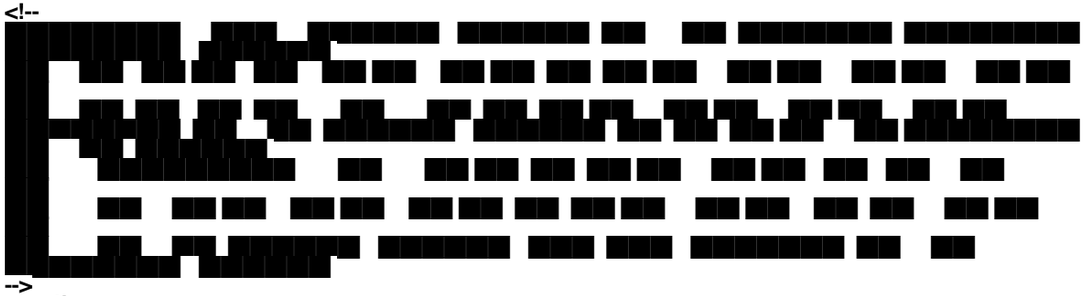
3. Run the following seps [=in parallel=]:

   1. Set |origin|'s <a>`prevent silent access` flag in the [=credential store=].

   2. [=Resolve=] |p| with `undefined`.

4. Retun |p|.

</section>

<!--

-->
<section>
Password Credentials # {#passwords}

For good or for ill, many websites rely on username/password pairs as an authentication mechanism. The {{PasswordCredential}} interface is a [=credential=] meant to enable this use case, storing both a username and password, as well as metadata that can help a user choose the right account from within a [=credential chooser=].

Examples ## {#password-examples}

8. Run the following substeps [=in parallel=]:

   1. Let |rl| be the result of executing |interfaces|[0]'s {{Credential/[[Create]](origin, options)}} internal method given |origin| and |options|.

   2. If |rl| is an [=exception=] or `null`, [=reject=] |p| with |rl|, and terminate these substeps.

   3. If |rl| is a {{Credential}}, [=resolve=] |p| with |rl|, and terminate these substeps.

   4. Assert: |rl| is a <code>[=constructCredential=]</code> algorithm.

   5. [=Queue a task=] on |global|'s [=DOM manipulation task source=] to run the following substeps:

      1. Let |result| be the result of [=promise-calling=] |rl| given |global|.
      2. If |result| is an [=exception=], [=reject=] |p| with |result| and terminate these substeps.
      3. [=Resolve=] |p| with |result|.

9. Return |p|.

<h4 id="algorithm-prevent-silent-access" algorithm>Prevent Silent Access</h4>

The <dfn abstract-op>Prevent Silent Access</dfn> algorithm accepts an [=environment settings object=] (|settings|), and returns a {{Promise}} which resolves once the `prevent silent access` flag is persisted to the [=credential store=].

<ol class="algorithm">
1. Let |origin| be |settings|' [=environment settings object/origin=].

2. Let |p| be [=a new promise=]

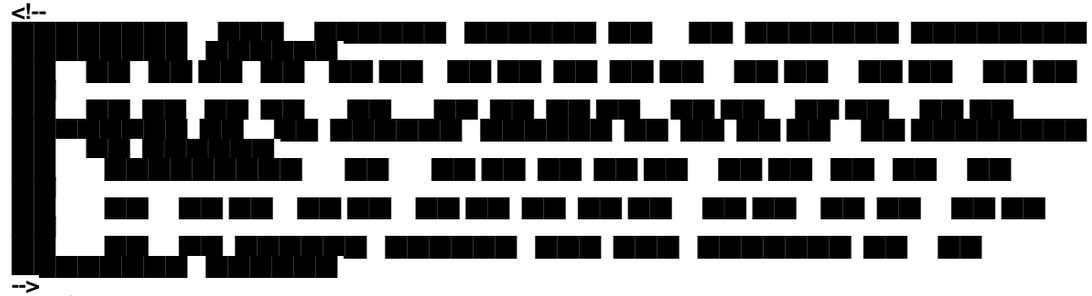
3. Run the following seps [=in parallel=]:

   1. Set |origin|'s <a>`prevent silent access` flag in the [=credential store=].

   2. [=Resolve=] |p| with `undefined`.

4. Retun |p|.

</section>

<!--

-->
<section>
Password Credentials # {#passwords}

For good or for ill, many websites rely on username/password pairs as an authentication mechanism. The {{PasswordCredential}} interface is a [=credential=] meant to enable this use case, storing both a username and password, as well as metadata that can help a user choose the right account from within a [=credential chooser=].

Examples ## {#password-examples}

```
### Password-based Sign-in ### {#examples-password-signin}

<div class="example">
  MegaCorp, Inc. supports passwords, and can use {{get()|navigator.credentials.get()}} to obtain
  username/password pairs from a user's [=credential store=]:

  <pre>
    navigator.<a attribute>credentials</a>
      .<a idl lt="get()" for="CredentialsContainer">get</a>({ '<a for="CredentialRequestOptions" dict-member>password</a>': true })
      .then(credential =&gt; {
        if (!credential) {
          // The user either doesn't have credentials for this site, or
          // refused to share them. Insert some code here to fall back to
          // a basic login form.
          return;
        }
        if (credential.<a attribute for="Credential">type</a> == '<a const>password</a>') {
          var form = new FormData();
          form.append('username_field', credential.id);
          form.append('password_field', credential.password);
          var opt = {
            method: 'POST',
            body: form,
            credentials: 'include'  // Send cookies.
          };
          fetch('https://example.com/loginEndpoint', opt)
            .then(function (response) {
              if (/* |response| indicates a successful login */) {
                // Record that the credential was effective. See note below.
                navigator.<a attribute>credentials</a>.<a idl lt="store()" for="CredentialsContainer">store</a>(credential);
                // Notify the user that sign-in succeeded! Do amazing, signed-in things!
                // Maybe navigate to a landing page via location.href =
                // '/signed-in-experience'?
              } else {
                // Insert some code here to fall back to a basic login form.
              }
            });
        }
      });
  </pre>

  Alternatively, the website could just copy the credential data into a <{form}> and call
  {{HTMLFormElement/submit()}} on the form:

  <pre>
    navigator.<a attribute>credentials</a>
      .<a idl lt="get()" for="CredentialsContainer">get</a>({ '<a for="CredentialRequestOptions" dict-member>password</a>': true })
      .then(credential =&gt; {
        if (!credential) {
          return; // as above...
        }
        if (credential.<a attribute for="Credential">type</a> == '<a const href="#password-literal">password</a>') {
          document.getQuerySelector('input[name=username_field]').value =
              credential.id;
          document.getQuerySelector('input[name=password_field]').value =
              credential.password;
          document.getQuerySelector('#myform').submit();
        }
      });
  </pre>

  Note that the former method is much preferred, as it contains an explicit call
  to {{CredentialsContainer/store()}} and saves the credentials. The <{form}> based mechanism
  relies on form submission, which navigates the browsing context, making it difficult to
  ensure that {{store()}} is called after successful sign-in.
```

```
    </div>

    Note: The [=credential chooser=] presented by the user agent could allow the user to choose
    credentials that aren't actually stored for the current origin. For instance, it might offer up
    credentials from `https://m.example.com` when signing into `https://www.example.com` (as
    described in [[#security-credential-access]]), or it might allow a user to create a new
    credential on the fly. Developers can deal gracefully with this uncertainty by calling
    {{CredentialsContainer/store()}} every time credentials are successfully used, even right after
    credentials have been retrieved from {{CredentialsContainer/get()}}: if the credentials aren't
    yet stored for the origin, the user will be given the opportunity to do so. If they are stored,
    the user won't be prompted.

    ### Post-sign-in Confirmation ### {#examples-post-signin}

    To ensure that users are offered to store new credentials after a successful sign-in, they can
    to be passed to {{CredentialsContainer/store()}}.

    <div class="example">
      If a user is signed in by submitting the credentials to a sign-in endpoint via {{fetch()}},
      we can check the response to determine whether the user
      was signed in successfully, and notify the user agent accordingly. Given a sign-in form like the
      following:

      <pre>
        &lt;form action="https://example.com/login" method="POST" id="theForm"&gt;
          &lt;label for="username"&gt;Username&lt;/label&gt;
          &lt;input type="text" id="username" name="username" <a element-attr
    for="input">autocomplete</a>="<a attr-value>username</a>"&gt;
          &lt;label for="password"&gt;Password&lt;/label&gt;
          &lt;input type="password" id="password" name="password" <a element-attr
    for="input">autocomplete</a>="<a attr-value>current-password</a>"&gt;
          &lt;input type="submit"&gt;
        &lt;/form&gt;
      </pre>

      Then the developer can handle the form submission with something like the following handler:

      <pre>
        document.querySelector('#theForm').addEventListener('submit', e =&gt; {
          if (<a attribute lt="credentials">navigator.credentials</a>) {
            e.preventDefault();

            // Construct a new <a idl>PasswordCredential</a> from the <a idl>HTMLFormElement</a>
            // that fired the "submit" event: this will suck up the values of the fields
            // labeled with "username" and "current-password" <a element-attr
    for="input">autocomplete</a>
            // attributes:
            var c = new <a idl lt="PasswordCredential(form)">PasswordCredential</a>(e.target);

            // Fetch the form's action URL, passing that new credential object in
            // as a FormData object. If the response indicates success, tell the user agent
            // so it can ask the user to store the password for future use:
            var opt = {
              method: 'POST',
              body: new FormData(e.target),
              credentials: 'include'  // Send cookies.
            };
            fetch(e.target.action, opt).then(r =&gt; {
              if (/* |r| is a "successful" <a idl>Response</a> */)
                <a idl lt="store()">navigator.credentials.store</a>(c);
            });
          }
        });
      </pre>
    </div>

    ### Change Password ### {#examples-change-password}

    This same storage mechanism can be reused for "password change" with no modifications: if the
```



```
    </div>

    Note: The [=credential chooser=] presented by the user agent could allow the user to choose
    credentials that aren't actually stored for the current origin. For instance, it might offer up
    credentials from `https://m.example.com` when signing into `https://www.example.com` (as
    described in [[#security-credential-access]]), or it might allow a user to create a new
    credential on the fly. Developers can deal gracefully with this uncertainty by calling
    {{CredentialsContainer/store()}} every time credentials are successfully used, even right after
    credentials have been retrieved from {{CredentialsContainer/get()}}: if the credentials aren't
    yet stored for the origin, the user will be given the opportunity to do so. If they are stored,
    the user won't be prompted.

    ### Post-sign-in Confirmation ### {#examples-post-signin}

    To ensure that users are offered to store new credentials after a successful sign-in, they can
    to be passed to {{CredentialsContainer/store()}}.

    <div class="example">
      If a user is signed in by submitting the credentials to a sign-in endpoint via {{fetch()}},
      we can check the response to determine whether the user
      was signed in successfully, and notify the user agent accordingly. Given a sign-in form like the
      following:

      <pre>
        &lt;form action="https://example.com/login" method="POST" id="theForm"&gt;
          &lt;label for="username"&gt;Username&lt;/label&gt;
          &lt;input type="text" id="username" name="username" <a element-attr
    for="input">autocomplete</a>="<a attr-value>username</a>"&gt;
          &lt;label for="password"&gt;Password&lt;/label&gt;
          &lt;input type="password" id="password" name="password" <a element-attr
    for="input">autocomplete</a>="<a attr-value>current-password</a>"&gt;
          &lt;input type="submit"&gt;
        &lt;/form&gt;
      </pre>

      Then the developer can handle the form submission with something like the following handler:

      <pre>
        document.querySelector('#theForm').addEventListener('submit', e =&gt; {
          if (<a attribute lt="credentials">navigator.credentials</a>) {
            e.preventDefault();

            // Construct a new <a idl>PasswordCredential</a> from the <a idl>HTMLFormElement</a>
            // that fired the "submit" event: this will suck up the values of the fields
            // labeled with "username" and "current-password" <a element-attr
    for="input">autocomplete</a>
            // attributes:
            var c = new <a idl lt="PasswordCredential(form)">PasswordCredential</a>(e.target);

            // Fetch the form's action URL, passing that new credential object in
            // as a FormData object. If the response indicates success, tell the user agent
            // so it can ask the user to store the password for future use:
            var opt = {
              method: 'POST',
              body: new FormData(e.target),
              credentials: 'include'  // Send cookies.
            };
            fetch(e.target.action, opt).then(r =&gt; {
              if (/* |r| is a "successful" <a idl>Response</a> */)
                <a idl lt="store()">navigator.credentials.store</a>(c);
            });
          }
        });
      </pre>
    </div>

    ### Change Password ### {#examples-change-password}

    This same storage mechanism can be reused for "password change" with no modifications: if the
```

```
    user changes their credentials, the website can notify the user agent that they've successfully
    signed in with new credentials. The user agent can then update the credentials it stores:

    <div class="example">
      MegaCorp Inc. allows users to change their passwords by POSTing data to
      a backend server asynchronously. After doing so successfully, they can
      update the user's credentials by calling {{CredentialsContainer/store()}}
      with the new information.

      Given a password change form like the following:

      <pre>
        &lt;form action="https://example.com/changePassword" method="POST" id="theForm"&gt;
          &lt;input type="hidden" name="username" <a element-attr for="input">autocomplete</a>="<a attr-value>username</a>" value="user"&gt;
          &lt;label for="password"&gt;New Password&lt;/label&gt;
          &lt;input type="password" id="password" name="password" <a element-attr for="input">autocomplete</a>="<a attr-value>new-password</a>"&gt;
          &lt;input type="submit"&gt;
        &lt;/form&gt;
      </pre>

      The developer can handle the form submission with something like the following:

      <pre>
        document.querySelector('#theForm').addEventListener('submit', e =&gt; {
          if (<a attribute lt="credentials">navigator.credentials</a>) {
            e.preventDefault();

            // Construct a new <a idl>PasswordCredential</a> from the <a idl>HTMLFormElement</a>
            // that fired the "submit" event: this will suck up the values of the fields
            // labeled with "username" and "new-password" <a element-attr for="input">autocomplete</a>
            // attributes:
            var c = new <a idl lt="PasswordCredential(form)">PasswordCredential</a>(e.target);

            // Fetch the form's action URL, passing that new credential object in
            // as a FormData object. If the response indicates success, tell the user agent
            // so it can ask the user to store the password for future use:
            var opt = {
              method: 'POST',
              body: new FormData(e.target),
              credentials: 'include'  // Send cookies.
            };
            fetch(e.target.action, opt).then(r =&gt; {
              if (/* |r| is a "successful" <a idl>Response</a> */)
                <a idl lt="store()">navigator.credentials.store</a>(c);
            });
          }
        });
      </pre>
    </div>

    ## The `PasswordCredential` Interface ## {#passwordcredential-interface}

    <pre class="idl">
      typedef (FormData or URLSearchParams) CredentialBodyType;

      [Constructor(HTMLFormElement form),
       Constructor(PasswordCredentialData data),
       Exposed=Window,
       SecureContext]
      interface PasswordCredential : Credential {
        readonly attribute USVString password;
      };
      PasswordCredential implements CredentialUserData;

      partial dictionary CredentialRequestOptions {
        boolean password = false;
      };
```


```
    user changes their credentials, the website can notify the user agent that they've successfully
    signed in with new credentials. The user agent can then update the credentials it stores:

    <div class="example">
      MegaCorp Inc. allows users to change their passwords by POSTing data to
      a backend server asynchronously. After doing so successfully, they can
      update the user's credentials by calling {{CredentialsContainer/store()}}
      with the new information.

      Given a password change form like the following:

      <pre>
        &lt;form action="https://example.com/changePassword" method="POST" id="theForm"&gt;
          &lt;input type="hidden" name="username" <a element-attr for="input">autocomplete</a>="<a attr-value>username</a>" value="user"&gt;
          &lt;label for="password"&gt;New Password&lt;/label&gt;
          &lt;input type="password" id="password" name="password" <a element-attr for="input">autocomplete</a>="<a attr-value>new-password</a>"&gt;
          &lt;input type="submit"&gt;
        &lt;/form&gt;
      </pre>

      The developer can handle the form submission with something like the following:

      <pre>
        document.querySelector('#theForm').addEventListener('submit', e =&gt; {
          if (<a attribute lt="credentials">navigator.credentials</a>) {
            e.preventDefault();

            // Construct a new <a idl>PasswordCredential</a> from the <a idl>HTMLFormElement</a>
            // that fired the "submit" event: this will suck up the values of the fields
            // labeled with "username" and "new-password" <a element-attr for="input">autocomplete</a>
            // attributes:
            var c = new <a idl lt="PasswordCredential(form)">PasswordCredential</a>(e.target);

            // Fetch the form's action URL, passing that new credential object in
            // as a FormData object. If the response indicates success, tell the user agent
            // so it can ask the user to store the password for future use:
            var opt = {
              method: 'POST',
              body: new FormData(e.target),
              credentials: 'include'  // Send cookies.
            };
            fetch(e.target.action, opt).then(r =&gt; {
              if (/* |r| is a "successful" <a idl>Response</a> */)
                <a idl lt="store()">navigator.credentials.store</a>(c);
            });
          }
        });
      </pre>
    </div>

    ## The `PasswordCredential` Interface ## {#passwordcredential-interface}

    <pre class="idl">
      typedef (FormData or URLSearchParams) CredentialBodyType;

      [Constructor(HTMLFormElement form),
       Constructor(PasswordCredentialData data),
       Exposed=Window,
       SecureContext]
      interface PasswordCredential : Credential {
        readonly attribute USVString password;
      };
      PasswordCredential implements CredentialUserData;

      partial dictionary CredentialRequestOptions {
        boolean password = false;
      };
```

```
    </pre>
    <dl dfn-for="PasswordCredential">
      :  <dfn attribute>password</dfn>
      ::  This attribute represents the password of the credential.

      :  {{Credential/[[type]]}}
      ::  The {{PasswordCredential}} [=interface object=] has an internal slot named `[[type]]` whose
          value is "<dfn const for="Credential/[[type]]">`password`</dfn>".

      :  {{Credential/[[discovery]]}}
      ::  The {{PasswordCredential}} [=interface object=] has an internal slot named `[[discovery]]`
          whose value is "{{Credential/[[discovery]]/credential store}}".

      :  <dfn constructor>PasswordCredential(form)</dfn>
      ::  This constructor accepts an {{HTMLFormElement}} (|form|), and runs the following steps:

          1.  Let |rl| be the result of executing <a abstract-op>Create a `PasswordCredential` from
              an `HTMLFormElement`</a> on |form|.

          2.  If |rl| is an [=exception=], [=throw=] |rl|.

              Otherwise, return |rl|.

      :  <dfn constructor>PasswordCredential(data)</dfn>
      ::  This constructor accepts a {{PasswordCredentialData}} (|data|), and runs the following steps:

          1.  Let |rl| be the result of executing <a abstract-op>Create a `PasswordCredential` from
              `PasswordCredentialData`</a> on |data|.

          2.  If |rl| is an [=exception=], [=throw=] |rl|.

              Otherwise, return |rl|.
    </dl>

    {{PasswordCredential}} objects can be created via
    {{CredentialsContainer/create()|navigator.credentials.create()}}
    either explicitly by passing in a {{PasswordCredentialData}} dictionary, or based on the contents
    of an {{HTMLFormElement}}'s [=submittable elements=].

    <pre class="idl">
      dictionary PasswordCredentialData : CredentialData {
        USVString name;
        USVString iconURL;
        required USVString password;
      };

      typedef (PasswordCredentialData or HTMLFormElement) PasswordCredentialInit;

      partial dictionary CredentialCreationOptions {
        PasswordCredentialInit password;
      };
    </pre>

    {{PasswordCredential}} objects are [=Credential/origin bound=].

    {{PasswordCredential}}'s [=interface object=] inherits {{Credential}}'s implementation of
    {{Credential/[[DiscoverFromExternalSource]](origin, options)}}, and defines its own implementation of
    {{PasswordCredential/[[CollectFromCredentialStore]](origin, options)}},
    {{PasswordCredential/[[Create]](options)}}, and
    {{PasswordCredential/[[Store]](credential)}}.

    ## Algorithms ## {#passwordcredential-algorithms}

    <h4 algorithm id="collectfromcredentialstore-passwordcredential">
      `PasswordCredential`'s `[[CollectFromCredentialStore]](origin, options)`
    </h4>

    <dfn for="PasswordCredential" method>\[[CollectFromCredentialStore]](origin, options)</dfn> is
  called
```

```
    </pre>
    <dl dfn-for="PasswordCredential">
      :  <dfn attribute>password</dfn>
      ::  This attribute represents the password of the credential.

      :  {{Credential/[[type]]}}
      ::  The {{PasswordCredential}} [=interface object=] has an internal slot named `[[type]]` whose
          value is "<dfn const for="Credential/[[type]]">`password`</dfn>".

      :  {{Credential/[[discovery]]}}
      ::  The {{PasswordCredential}} [=interface object=] has an internal slot named `[[discovery]]`
          whose value is "{{Credential/[[discovery]]/credential store}}".

      :  <dfn constructor>PasswordCredential(form)</dfn>
      ::  This constructor accepts an {{HTMLFormElement}} (|form|), and runs the following steps:

          1.  Let |rl| be the result of executing <a abstract-op>Create a `PasswordCredential` from
              an `HTMLFormElement`</a> on |form|.

          2.  If |rl| is an [=exception=], [=throw=] |rl|.

              Otherwise, return |rl|.

      :  <dfn constructor>PasswordCredential(data)</dfn>
      ::  This constructor accepts a {{PasswordCredentialData}} (|data|), and runs the following steps:

          1.  Let |rl| be the result of executing <a abstract-op>Create a `PasswordCredential` from
              `PasswordCredentialData`</a> on |data|.

          2.  If |rl| is an [=exception=], [=throw=] |rl|.

              Otherwise, return |rl|.
    </dl>

    {{PasswordCredential}} objects can be created via
    {{CredentialsContainer/create()|navigator.credentials.create()}}
    either explicitly by passing in a {{PasswordCredentialData}} dictionary, or based on the contents
    of an {{HTMLFormElement}}'s [=submittable elements=].

    <pre class="idl">
      dictionary PasswordCredentialData : CredentialData {
        USVString name;
        USVString iconURL;
        required USVString password;
      };

      typedef (PasswordCredentialData or HTMLFormElement) PasswordCredentialInit;

      partial dictionary CredentialCreationOptions {
        PasswordCredentialInit password;
      };
    </pre>

    {{PasswordCredential}} objects are [=Credential/origin bound=].

    {{PasswordCredential}}'s [=interface object=] inherits {{Credential}}'s implementation of
    {{Credential/[[DiscoverFromExternalSource]](origin, options)}}, and defines its own implementation of
    {{PasswordCredential/[[CollectFromCredentialStore]](origin, options)}},
    {{PasswordCredential/[[Create]](origin, options)}}, and
    {{PasswordCredential/[[Store]](credential)}}.

    ## Algorithms ## {#passwordcredential-algorithms}

    <h4 algorithm id="collectfromcredentialstore-passwordcredential">
      `PasswordCredential`'s `[[CollectFromCredentialStore]](origin, options)`
    </h4>

    <dfn for="PasswordCredential" method>\[[CollectFromCredentialStore]](origin, options)</dfn> is
  called
```

```
1153      with an [=environment settings object/origin=] (|origin|) and a {{CredentialRequestOptions}} (|
          options|),
1154      and returns a set of {{Credential}} objects from
1155      the [=credential store=]. If no matching {{Credential}} objects are available, the returned set
1156      will be empty.
1157
1158      <ol class="algorithm">
1159        1. Assert: |options|["{{CredentialRequestOptions/password}}"] [=map/exists=].
1160
1161        2. If |options|["{{CredentialRequestOptions/password}}"] is not `true`, return the empty set.
1162
1163        3. Return the result of <a abstract-op lt="Retrieve a list of credentials">retrieving</a>
1164           credentials from the [=credential store=] that match the following filter:
1165
1166           1. The credential is a {{PasswordCredential}}
1167           2. The credential's {{Credential/[[origin]]}} is the [=same origin=] as |origin|.
1168      </ol>
1169
1170      <h4 algorithm id="create-passwordcredential">
1171        `PasswordCredential`'s `[[Create]](options)`
1172      </h4>
1173
1174      <dfn for="PasswordCredential" method>\[[Create]](options)</dfn> is called with a
1175      {{CredentialCreationOptions}} (|options|), and returns a {{PasswordCredential}} if one can be created,
1176      `null` otherwise. The {{CredentialCreationOptions}} dictionary must have a `password` member which
1177      holds either an {{HTMLFormElement}} or a {{PasswordCredentialData}}. If that member's value cannot
          be
1178      used to create a {{PasswordCredential}}, this algorithm will return a {{TypeError}} [=exception=].
1179
1180      <ol class="algorithm">
1181        1. Assert: |options|["{{CredentialCreationOptions/password}}"] [=map/exists=].
1182
1183        2. If |options|["{{CredentialCreationOptions/password}}"] is an {{HTMLFormElement}}, return the
1184           result of executing <a abstract-op>Create a `PasswordCredential` from an
1185           `HTMLFormElement`</a> on |options|["{{CredentialCreationOptions/password}}"].
1186
1187        3. If |options|["{{CredentialCreationOptions/password}}"] is a {{PasswordCredentialData}}, return
1188           the result of executing <a abstract-op>Create a `PasswordCredential` from
1189           `PasswordCredentialData`</a> on |options|["{{CredentialCreationOptions/password}}"].
1190
1191        4. Return a {{TypeError}} [=exception=].
1192      </ol>
1193
1194      <h4 algorithm id="store-passwordcredential">
1195        `PasswordCredential`'s `[[Store]](credential)`
1196      </h4>
1197
1198      <dfn for="PasswordCredential" method>\[[Store]](credential)</dfn> is called with a
1199      {{PasswordCredential}} (|credential|), and returns once |credential| is persisted to the
1200      [=credential store=].
1201
1202      <ol class="algorithm">
1203        1. If the user agent's [=credential store=] contains a {{PasswordCredential}} (|stored|)
1204           whose {{Credential/id}} attribute is |credential|'s {{Credential/id}} and whose
1205           {{[[origin]]}} slot is the [=same origin=] as |credential|'s {{Credential/[[origin]]}},
1206           then:
1207
1208           1. If the user grants permission to update credentials (as discussed when defining
1209              [=user mediation=]), then:
1210
1211              1. Set |stored|'s <a attribute for="PasswordCredential">`password`</a> to |credential|'s
1212                 <a attribute for="PasswordCredential">`password`</a>.
1213
1214              2. Set |stored|'s {{CredentialUserData/name}} to |credential|'s
1215                 {{CredentialUserData/name}}.
1216
1217              3. Set |stored|'s {{CredentialUserData/iconURL}} to |credential|'s
1218                 {{CredentialUserData/iconURL}}.
```

```
1205      with an [=environment settings object/origin=] (|origin|) and a {{CredentialRequestOptions}} (|
          options|),
1206      and returns a set of {{Credential}} objects from
1207      the [=credential store=]. If no matching {{Credential}} objects are available, the returned set
1208      will be empty.
1209
1210      <ol class="algorithm">
1211        1. Assert: |options|["{{CredentialRequestOptions/password}}"] [=map/exists=].
1212
1213        2. If |options|["{{CredentialRequestOptions/password}}"] is not `true`, return the empty set.
1214
1215        3. Return the result of <a abstract-op lt="Retrieve a list of credentials">retrieving</a>
1216           credentials from the [=credential store=] that match the following filter:
1217
1218           1. The credential is a {{PasswordCredential}}
1219           2. The credential's {{Credential/[[origin]]}} is the [=same origin=] as |origin|.
1220      </ol>
1221
1222      <h4 algorithm id="create-passwordcredential">
1223        `PasswordCredential`'s `[[Create]](origin, options)`
1224      </h4>
1225
1226      <dfn for="PasswordCredential" method>\[[Create]](origin, options)</dfn> is called with
1227      an [=environment settings object/origin=] (<var ignore>origin</var>) and a
1228      {{CredentialCreationOptions}} (|options|), and returns a {{PasswordCredential}} if one can be created,
1229      `null` otherwise. The {{CredentialCreationOptions}} dictionary must have a `password` member which
1230      holds either an {{HTMLFormElement}} or a {{PasswordCredentialData}}. If that member's value cannot
          be
1231      used to create a {{PasswordCredential}}, this algorithm will return a {{TypeError}} [=exception=].
1232
1233      <ol class="algorithm">
1234        1. Assert: |options|["{{CredentialCreationOptions/password}}"] [=map/exists=].
1235
1236        2. If |options|["{{CredentialCreationOptions/password}}"] is an {{HTMLFormElement}}, return the
1237           result of executing <a abstract-op>Create a `PasswordCredential` from an
1238           `HTMLFormElement`</a> on |options|["{{CredentialCreationOptions/password}}"].
1239
1240        3. If |options|["{{CredentialCreationOptions/password}}"] is a {{PasswordCredentialData}}, return
1241           the result of executing <a abstract-op>Create a `PasswordCredential` from
1242           `PasswordCredentialData`</a> on |options|["{{CredentialCreationOptions/password}}"].
1243
1244        4. Return a {{TypeError}} [=exception=].
1245      </ol>
1246
1247      <h4 algorithm id="store-passwordcredential">
1248        `PasswordCredential`'s `[[Store]](credential)`
1249      </h4>
1250
1251      <dfn for="PasswordCredential" method>\[[Store]](credential)</dfn> is called with a
1252      {{PasswordCredential}} (|credential|), and returns once |credential| is persisted to the
1253      [=credential store=].
1254
1255      <ol class="algorithm">
1256        1. If the user agent's [=credential store=] contains a {{PasswordCredential}} (|stored|)
1257           whose {{Credential/id}} attribute is |credential|'s {{Credential/id}} and whose
1258           {{[[origin]]}} slot is the [=same origin=] as |credential|'s {{Credential/[[origin]]}},
1259           then:
1260
1261           1. If the user grants permission to update credentials (as discussed when defining
1262              [=user mediation=]), then:
1263
1264              1. Set |stored|'s <a attribute for="PasswordCredential">`password`</a> to |credential|'s
1265                 <a attribute for="PasswordCredential">`password`</a>.
1266
1267              2. Set |stored|'s {{CredentialUserData/name}} to |credential|'s
1268                 {{CredentialUserData/name}}.
1269
1270              3. Set |stored|'s {{CredentialUserData/iconURL}} to |credential|'s
1271                 {{CredentialUserData/iconURL}}.
```

```
Otherwise, if the user grants permission to store credentials (as discussed when
defining [=user mediation=], then:

1.  Store a {{PasswordCredential}} in the [=credential store=] with the following
    properties:

    :  {{Credential/id}}
    ::  |credential|'s {{Credential/id}}
    :  {{CredentialUserData/name}},
    ::  |credential|'s {{CredentialUserData/name}}
    :  {{CredentialUserData/iconURL}}
    ::  |credential|'s {{CredentialUserData/iconURL}}
    :  {{Credential/[[origin]]}}
    ::  |credential|'s {{Credential/[[origin]]}}
    :  <a attribute for="PasswordCredential">`password`</a>
    ::  |credential|'s <a attribute for="PasswordCredential">`password`</a>

</ol>

<h4 algorithm id="construct-passwordcredential-form">
  Create a `PasswordCredential` from an `HTMLFormElement`
</h4>

To <dfn abstract-op>Create a `PasswordCredential` from an `HTMLFormElement`</dfn>, given an
{{HTMLFormElement}} |form|, run these steps.

Note: [[#examples-post-signin]] and [[#examples-change-password]] provide examples of the
intended
usage.

<ol class="algorithm">
  1.  Let |data| be a new {{PasswordCredentialData}} dictionary.

  2.  Let |formData| be the result of executing the {{FormData}} constructor
      on |form|.

  3.  Let |elements| be a list of all the [=submittable elements=] whose [=form owner=] is |form|, in
[=tree order=].

  4.  Let |newPasswordObserved| be `false`.

  5.  For each |field| in |elements|, run the following steps:

      1.  If |field| does not have an <{input/autocomplete}> attribute, then skip to the next
          |field|.

      2.  Let |name| be the value of |field|'s <{input/name}> attribute.

      3.  If |formData|'s {{FormData/has()}} method returns `false` when executed on |name|, then
          skip to the next |field|.

      4.  If |field|'s <{input/autocomplete}> attribute's value contains one or more [=autofill
          detail tokens=] (|tokens|), then:

          1.  For each |token| in |tokens|:

              1.  If |token| is an <a>ASCII case-insensitive</a> match for one
                  of the following strings, run the associated steps:

                  :  "<a attr-value>`new-password`</a>"
                  ::  Set |data|'s {{PasswordCredentialData/password}} member's
                      value to the result of executing |formData|'s
                      {{FormData/get()}} method on |name|, and |newPasswordObserved| to `true`.

                  :  "<a attr-value>`current-password`</a>"
                  ::  If |newPasswordObserved| is `false`,
                      set |data|'s {{PasswordCredentialData/password}} member's
                      value to the result of executing |formData|'s
```

```
                    {{FormData/get()}} method on |name|.

                    Note: By checking that |newPasswordObserved| is `false`,
                    `new-password` fields take precedence over
                    `current-password` fields.

                :   "<a attr-value>`photo`</a>"
                ::  Set |data|'s {{CredentialUserData/iconURL}} member's
                    value to the result of executing |formData|'s
                    {{FormData/get()}} method on |name|.

                :   "<a attr-value>`name`</a>"
                :   "<a attr-value>`nickname`</a>"
                ::  Set |data|'s {{CredentialUserData/name}} member's
                    value to the result of executing |formData|'s
                    {{FormData/get()}} method on |name|.

                :   "<a attr-value>`username`</a>"
                ::  Set |data|'s {{CredentialData/id}} member's value to the
                    result of executing |formData|'s {{FormData/get()}} method
                    on |name|.

  6. Let |cl| be the result of executing <a abstract-op>Create a `PasswordCredential` from
     `PasswordCredentialData`</a> on |data|.

  7. If |cl| is an [=exception=], return |cl|.

  8. Assert: |cl| is a {{PasswordCredential}}.

  9. Return |cl|.
</ol>

<h4 algorithm id="construct-passwordcredential-data">
  Create a `PasswordCredential` from `PasswordCredentialData`
</h4>

To <dfn abstract-op>Create a `PasswordCredential` from `PasswordCredentialData`</dfn>, given an
{{PasswordCredentialData}} (|data|), run these steps.

<ol class="algorithm">
  1. Let |cl| be a new {{PasswordCredential}} object.

  2. If any of the following are the empty string, return a {{TypeError}} [=exception=]:

     * |data|'s {{CredentialData/id}} member's value
     * |data|'s {{PasswordCredentialData/password}} member's value

  3. Set |cl|'s properties as follows:

     :  <a attribute for="PasswordCredential">`password`</a>
     ::  |data|'s {{PasswordCredentialData/password}} member's value
     :  {{Credential/id}}
     ::  |data|'s {{CredentialData/id}} member's value
     :  {{CredentialUserData/iconURL}}
     ::  |data|'s {{PasswordCredentialData/iconURL}} member's value
     :  {{CredentialUserData/name}}
     ::  |data|'s {{PasswordCredentialData/name}} member's value
     :  {{Credential/[[origin]]}}
     ::  The [=environment settings object/origin=] of the [=current settings object=].

  4. Return |cl|.
</ol>

<h4 algorithm id="passwordcredential-matching">
  `CredentialRequestOptions` Matching for `PasswordCredential`
</h4>

Given a {{CredentialRequestOptions}} (|options|), the following algorithm returns "`Matches`" if
the {{PasswordCredential}} should be available as a response to a {{CredentialsContainer/get()}}
```

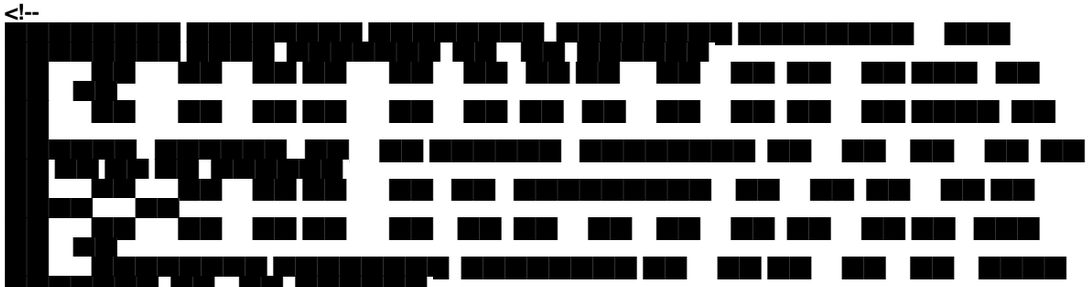
request, and "`Does Not Match`" otherwise.

1. If |options| has a {{CredentialRequestOptions/password}} member whose value is `true`, then return "`Matches`".

2. Return "`Does Not Match`".

</section>

<!--



-->

<section>

Federated Credentials # {#federated}

The `FederatedCredential` Interface ## {#federatedcredential-interface}

<pre class="idl">
[Constructor(FederatedCredentialInit data),
 Exposed=Window,
 SecureContext]
interface FederatedCredential : Credential {
  readonly attribute USVString provider;
  readonly attribute DOMString? protocol;
};
FederatedCredential implements CredentialUserData;

dictionary FederatedCredentialRequestOptions {
  sequence<USVString> providers;
  sequence<DOMString> protocols;
};

partial dictionary CredentialRequestOptions {
  FederatedCredentialRequestOptions federated;
};
</pre>

<dl dfn-for="FederatedCredential">
: <dfn attribute>provider</dfn>
:: The credential's federated identity provider. See [[#provider-identification]] for details regarding valid formats.

: <dfn attribute>protocol</dfn>
:: The credential's federated identity provider's protocol (e.g. "`openidconnect`"). If the value is `null`, then the protocol can be inferred from the {{FederatedCredential/provider}}.

: {{Credential/[[type]]}}
:: The {{FederatedCredential}} [=interface object=] has an internal slot named `[[type]]` whose value is "<dfn const>`federated`</dfn>".

: {{Credential/[[discovery]]}}
:: The {{FederatedCredential}} [=interface object=] has an internal slot named `[[discovery]]` whose value is "{{Credential/[[discovery]]/credential store}}".

: <dfn constructor>FederatedCredential(data)</dfn>
:: This constructor accepts a {{FederatedCredentialInit}} (|data|), and runs the following steps:

1. Let |rl| be the result of executing <a abstract-op>Create a `FederatedCredential` from `FederatedCredentialInit` on |data|.

2. If |rl| is an [=exception=], [=throw=] |rl|.

   Otherwise, return |rl|.

</dl>

{{FederatedCredential}} objects can be created by passing a {{FederatedCredentialInit}} dictionary into {{CredentialsContainer/create()|navigator.credentials.create()}}.

<pre class="idl">
dictionary FederatedCredentialInit : CredentialData {
  USVString name;
  USVString iconURL;
  required USVString provider;
  DOMString protocol;
};

partial dictionary CredentialCreationOptions {
  FederatedCredentialInit federated;
};
</pre>

{{FederatedCredential}} objects are [=Credential/origin bound=].

{{FederatedCredential}}'s [=interface object=] inherits {{Credential}}'s implementation of {{Credential/[[DiscoverFromExternalSource]](origin, options)}}, and defines its own implementation of {{FederatedCredential/[[CollectFromCredentialStore]](origin, options)}}, {{FederatedCredential/[[Create]](options)}}, and {{FederatedCredential/[[Store]](credential)}}.

Note: If, in the future, we teach the user agent to obtain authentication tokens on a user's behalf, we could do so by building an implementation of `[[DiscoverFromExternalSource]](origin, options)`.

Identifying Providers ### {#provider-identification}

Every site should use the same identifier when referring to a specific federated identity provider. For example, Facebook Login shouldn't be referred to as "Facebook" and "Facebook Login" and "FB" and "FBL" and "Facebook.com" and so on. It should have a canonical identifier which everyone can make use of, as consistent identification makes it possible for user agents to be helpful.

For consistency, federations passed into the APIs defined in this document (e.g. {{FederatedCredentialRequestOptions}}'s {{FederatedCredentialRequestOptions/providers}} array, or {{FederatedCredential}}'s {{FederatedCredential/provider}} property) MUST be identified by the ASCII serialization of the origin the provider uses for sign in. That is, Facebook would be represented by `https://www.facebook.com` and Google by `https://accounts.google.com`.

This serialization of an [=origin=] does _not_ include a trailing U+002F SOLIDUS ("`/`"), but user agents SHOULD accept them silently: `https://accounts.google.com/` is clearly intended to be the same as `https://accounts.google.com`.

Algorithms ## {#federatedcredential-algorithms}

<h4 algorithm id="collectfromcredentialstore-federatedcredential">
`FederatedCredential`'s `[[CollectFromCredentialStore]](origin, options)`
</h4>

<dfn for="FederatedCredential" method>\[[CollectFromCredentialStore]](origin, options)</dfn> is called with an [=environment settings object/origin=] (|origin|) and a {{CredentialRequestOptions}} (|options|), and returns a set of {{Credential}} objects from

```
1482      the [=credential store=]. If no matching {{Credential}} objects are available, the returned set
1483      will be empty.
1484
1485      <ol class="algorithm">
1486        1.  Assert: !options!["{{CredentialRequestOptions/federated}}"] [=map/exists=].
1487
1488        2.  Return the result of <a abstract-op lt="Retrieve a list of credentials">retrieving</a>
1489            credentials from the [=credential store=] that match the following filter:
1490
1491              1.  The credential is a {{FederatedCredential}}
1492              2.  The credential's {{Credential/[[origin]]}} is the [=same origin=] as !origin!.
1493              3.  If !options!["{{CredentialRequestOptions/federated}}"]
                  ["{{FederatedCredentialRequestOptions/providers}}"]
1494                  [=map/exists=], its value [=list/contains=] the credentials's {{FederatedCredential/provider}}.
1495              4.  If !options!["{{CredentialRequestOptions/federated}}"]
                  ["{{FederatedCredentialRequestOptions/protocols}}"]
1496                  [=map/exists=], its value [=list/contains=] the credentials's {{FederatedCredential/protocol}}.
1497        </ol>
1498
1499      <h4 algorithm id="create-federatedcredential">
1500        `FederatedCredential`'s `[[Create]](options)`
1501      </h4>
1502
1503      <dfn for="FederatedCredential" method>\[[Create]](options)</dfn> is called with a
1504      {{CredentialCreationOptions}} (!options!), and returns a {{FederatedCredential}} if one can be created,
1505      `null` otherwise, or an [=exception=] in exceptional circumstances:
1506
1507      <ol class="algorithm">
1508        1.  Assert: !options!["{{CredentialCreationOptions/federated}}"] [=map/exists=].
1509
1510        2.  Return the result of executing <a abstract-op>Create a `FederatedCredential` from
1511            `FederatedCredentialInit`</a> on !options!["{{CredentialCreationOptions/federated}}"].
1512      </ol>
1513
1514      <h4 algorithm id="store-federatedcredential">
1515        `FederatedCredential`'s `[[Store]](credential)`
1516      </h4>
1517
1518      <dfn for="FederatedCredential" method>\[[Store]](credential)</dfn> is called with a
1519      {{FederatedCredential}} (!credential!), and returns once !credential! is persisted to the
1520      [=credential store=].
1521
1522      <ol class="algorithm">
1523        1.  If the user agent's [=credential store=] contains a {{FederatedCredential}} whose
1524            {{Credential/id}} attribute is !credential!'s {{Credential/id}} and whose {{[[origin]]}}
1525            slot is the [=same origin=] as !credential!'s {{Credential/[[origin]]}}, and
1526            whose {{FederatedCredential/provider}} is !credential!'s
1527            {{FederatedCredential/provider}}, then return.
1528
1529        2.  Store a {{FederatedCredential}} in the [=credential store=] with the following
1530            properties:
1531
1532            :  {{Credential/id}}
1533            ::  !credential!'s {{Credential/id}}
1534            :  {{CredentialUserData/name}},
1535            ::  !credential!'s {{CredentialUserData/name}}
1536            :  {{CredentialUserData/iconURL}}
1537            ::  !credential!'s {{CredentialUserData/iconURL}}
1538            :  {{Credential/[[origin]]}}
1539            ::  !credential!'s {{Credential/[[origin]]}}
1540            :  {{FederatedCredential/provider}}
1541            ::  !credential!'s {{FederatedCredential/provider}}
1542            :  {{FederatedCredential/protocol}}
1543            ::  !credential!'s {{FederatedCredential/protocol}}
1544      </ol>
1545
1546      <h4 algorithm id="construct-federatedcredential-data">
1547        Create a `FederatedCredential` from `FederatedCredentialInit`
```

```
1535      the [=credential store=]. If no matching {{Credential}} objects are available, the returned set
1536      will be empty.
1537
1538      <ol class="algorithm">
1539        1.  Assert: !options!["{{CredentialRequestOptions/federated}}"] [=map/exists=].
1540
1541        2.  Return the result of <a abstract-op lt="Retrieve a list of credentials">retrieving</a>
1542            credentials from the [=credential store=] that match the following filter:
1543
1544              1.  The credential is a {{FederatedCredential}}
1545              2.  The credential's {{Credential/[[origin]]}} is the [=same origin=] as !origin!.
1546              3.  If !options!["{{CredentialRequestOptions/federated}}"]
                  ["{{FederatedCredentialRequestOptions/providers}}"]
1547                  [=map/exists=], its value [=list/contains=] the credentials's {{FederatedCredential/provider}}.
1548              4.  If !options!["{{CredentialRequestOptions/federated}}"]
                  ["{{FederatedCredentialRequestOptions/protocols}}"]
1549                  [=map/exists=], its value [=list/contains=] the credentials's {{FederatedCredential/protocol}}.
1550        </ol>
1551
1552      <h4 algorithm id="create-federatedcredential">
1553        `FederatedCredential`'s `[[Create]](origin, options)`
1554      </h4>
1555
1556      <dfn for="FederatedCredential" method>\[[Create]](origin, options)</dfn> is called with
1557      an [=environment settings object/origin=] (<var ignore>origin</var>) and a
1558      {{CredentialCreationOptions}} (!options!), and returns a {{FederatedCredential}} if one can be created,
1559      `null` otherwise, or an [=exception=] in exceptional circumstances:
1560
1561      <ol class="algorithm">
1562        1.  Assert: !options!["{{CredentialCreationOptions/federated}}"] [=map/exists=].
1563
1564        2.  Return the result of executing <a abstract-op>Create a `FederatedCredential` from
1565            `FederatedCredentialInit`</a> on !options!["{{CredentialCreationOptions/federated}}"].
1566      </ol>
1567
1568      <h4 algorithm id="store-federatedcredential">
1569        `FederatedCredential`'s `[[Store]](credential)`
1570      </h4>
1571
1572      <dfn for="FederatedCredential" method>\[[Store]](credential)</dfn> is called with a
1573      {{FederatedCredential}} (!credential!), and returns once !credential! is persisted to the
1574      [=credential store=].
1575
1576      <ol class="algorithm">
1577        1.  If the user agent's [=credential store=] contains a {{FederatedCredential}} whose
1578            {{Credential/id}} attribute is !credential!'s {{Credential/id}} and whose {{[[origin]]}}
1579            slot is the [=same origin=] as !credential!'s {{Credential/[[origin]]}}, and
1580            whose {{FederatedCredential/provider}} is !credential!'s
1581            {{FederatedCredential/provider}}, then return.
1582
1583        2.  Store a {{FederatedCredential}} in the [=credential store=] with the following
1584            properties:
1585
1586            :  {{Credential/id}}
1587            ::  !credential!'s {{Credential/id}}
1588            :  {{CredentialUserData/name}},
1589            ::  !credential!'s {{CredentialUserData/name}}
1590            :  {{CredentialUserData/iconURL}}
1591            ::  !credential!'s {{CredentialUserData/iconURL}}
1592            :  {{Credential/[[origin]]}}
1593            ::  !credential!'s {{Credential/[[origin]]}}
1594            :  {{FederatedCredential/provider}}
1595            ::  !credential!'s {{FederatedCredential/provider}}
1596            :  {{FederatedCredential/protocol}}
1597            ::  !credential!'s {{FederatedCredential/protocol}}
1598      </ol>
1599
1600      <h4 algorithm id="construct-federatedcredential-data">
1601        Create a `FederatedCredential` from `FederatedCredentialInit`
```

</h4>

To <dfn abstract-op>Create a `FederatedCredential` from `FederatedCredentialInit`</dfn>, given a {{FederatedCredentialInit}} (*init*), run these steps.

<ol class="algorithm">
1. Let *c* be a new {{FederatedCredential}} object.

2. If any of the following are the empty string, return a {{TypeError}} [=exception=]:

    * *init*.{{CredentialData/id}}'s value
    * *init*.{{FederatedCredentialInit/provider}}'s value

3. Set *c*'s properties as follows:

    : {{Credential/id}}
    :: *init*.{{CredentialData/id}}'s value
    : {{FederatedCredential/provider}}
    :: *init*.{{FederatedCredentialInit/provider}}'s value
    : {{CredentialUserData/iconURL}}
    :: *init*.{{CredentialUserData/iconURL}}'s value
    : {{CredentialUserData/name}}
    :: *init*.{{CredentialUserData/name}}'s value
    : {{Credential/[[origin]]}}
    :: The [=environment settings object/origin=] of the [=current settings object=].

4. Return *c*.

</section>

<!--

-->
<section>
User Mediation # {#user-mediation}

Exposing credential information to the web via an API has a number of potential impacts on user privacy. The user agent, therefore, MUST involve the user in a number of cases in order to ensure that they clearly understands what's going on, and with whom their credentials are being shared.

We call a particular action <dfn export lt="user mediated|user mediation">user mediated</dfn> if it takes place after gaining a user's explicit consent. Consent might be expressed through a user's direct interaction with a [=credential chooser=] interface, for example. In general, [=user mediated=] actions will involve presenting the user some sort of UI, and asking them to make a decision.

An action is unmediated if it takes place silently, without explicit user consent. For example, if a user configures their browser to grant persistent credential access to a particular origin, credentials may be provided without presenting the user with a UI requesting a decision.

Here we'll spell out a few requirements that hold for all credential types, but note that there's a good deal of latitude left up to the user agent (which is in a priviliged position to assist the user). Moreover, specific credential types may have distinct requirements that exceed the requirements laid out more generally here.

Storing and Updating Credentials ## {#user-mediated-storage}

Credential information is sensitive data, and users MUST remain in control of that information's storage. Inadvertent credential storage could, for instance, unexpectedly link a user's local

</h4>

To <dfn abstract-op>Create a `FederatedCredential` from `FederatedCredentialInit`</dfn>, given a {{FederatedCredentialInit}} (*init*), run these steps.

<ol class="algorithm">
1. Let *c* be a new {{FederatedCredential}} object.

2. If any of the following are the empty string, return a {{TypeError}} [=exception=]:

    * *init*.{{CredentialData/id}}'s value
    * *init*.{{FederatedCredentialInit/provider}}'s value

3. Set *c*'s properties as follows:

    : {{Credential/id}}
    :: *init*.{{CredentialData/id}}'s value
    : {{FederatedCredential/provider}}
    :: *init*.{{FederatedCredentialInit/provider}}'s value
    : {{CredentialUserData/iconURL}}
    :: *init*.{{CredentialUserData/iconURL}}'s value
    : {{CredentialUserData/name}}
    :: *init*.{{CredentialUserData/name}}'s value
    : {{Credential/[[origin]]}}
    :: The [=environment settings object/origin=] of the [=current settings object=].

4. Return *c*.

</section>

<!--

-->
<section>
User Mediation # {#user-mediation}

Exposing credential information to the web via an API has a number of potential impacts on user privacy. The user agent, therefore, MUST involve the user in a number of cases in order to ensure that they clearly understands what's going on, and with whom their credentials are being shared.

We call a particular action <dfn export lt="user mediated|user mediation">user mediated</dfn> if it takes place after gaining a user's explicit consent. Consent might be expressed through a user's direct interaction with a [=credential chooser=] interface, for example. In general, [=user mediated=] actions will involve presenting the user some sort of UI, and asking them to make a decision.

An action is unmediated if it takes place silently, without explicit user consent. For example, if a user configures their browser to grant persistent credential access to a particular origin, credentials may be provided without presenting the user with a UI requesting a decision.

Here we'll spell out a few requirements that hold for all credential types, but note that there's a good deal of latitude left up to the user agent (which is in a priviliged position to assist the user). Moreover, specific credential types may have distinct requirements that exceed the requirements laid out more generally here.

Storing and Updating Credentials ## {#user-mediated-storage}

Credential information is sensitive data, and users MUST remain in control of that information's storage. Inadvertent credential storage could, for instance, unexpectedly link a user's local

profile on a particular device to a specific online persona. To mitigate the risk of surprise:

1. Credential information SHOULD NOT be stored or updated without [=user mediation=]. For example, the user agent could display a "Save this credential?" dialog box to the user in response to each call to {{store()}}.

   User consent MAY be inferred if a user agent chooses to offer a persistant grant of consent in the form of an "Always save passwords" option (though we'd suggest that user agents should err on the side of something more narrowly scoped: perhaps "Always save _generated_ passwords.", or "Always save passwords for this site.").

2. User agents SHOULD notify users when credentials are stored. This might take the form of an icon in the address bar, or some similar location.

3. User agents MUST allow users to manually remove stored credentials. This functionality might be implemented as a settings page, or via interaction with a notification as described above.

Requiring User Mediation ## {#user-mediation-requirement}

By default, [=user mediation=] is required for all [=origins=], as the relevant [=prevent silent access flag=] in the [=credential store=] is set to `true`. Users MAY choose to grant an origin persistent access to credentials (perhaps in the form of a "Stay signed into this site." option), which would set this flag to `false`. In this case, the user would always be signed into that site, which is desirable from the perspective of usability and convinience, but which might nevertheless have surprising implications (consider a user agent which syncs this flag's state across devices, for instance).

To mitigate the risk of surprise:

1. User agents MUST allow users to require [=user mediation=] for a given origin or for all origins. This functionality might be implemented as a global toggle that overrides each origin's <a>`prevent silent access` flag to return `false`, or via more granular settings for specific origins (or specific credentials on specific origins).

2. User agents MUST NOT set an [=origin=]'s <a>`prevent silent access` flag to `false` without [=user mediation=]. For example, the [=credential chooser=] described in [[#user-mediated-selection]] could have a checkbox which the user could toggle to mark a credential as available without mediation for the origin, or the user agent could have an onboarding process for its credential manager which asked a user for a default setting.

3. User agents MUST notify users when credentials are provided to an origin. This could take the form of an icon in the address bar, or some similar location.

4. If a user clears her browsing data for an origin (cookies, localStorage, and so on), the user agent MUST set the <a>`prevent silent access` flag to `true` for that origin.

Credential Selection ## {#user-mediated-selection}

When responding to a call to {{CredentialsContainer/get()}} on an origin which requires [=user mediation=], user agents MUST ask the user for permission to share credential information. This SHOULD take the form of a <dfn export>credential chooser</dfn> which presents the user with a list of credentials that are available for use on a site, allowing them to select one which should be provided to the website, or to reject the request entirely.

The chooser interface SHOULD be implemented in such a way as to be distinguishable from UI which a website could produce. For example, the chooser might overlap the user agent's UI in some unspoofable way.

The chooser interface MUST include an indication of the origin which is requesting credentials.

The chooser interface SHOULD include all {{Credential}} objects associated with the origin that requested credentials.

User agents MAY internally associate information with each {{Credential}} object beyond the attributes specified in this document in order to enhance the utility of such a chooser. For example, favicons could help disambiguate identity providers, etc. Any additional information stored MUST not be exposed directly to the web.

The chooser's behavior is not defined here: user agents are encouraged to experiment with UI treatments that educate users about their authentication options, and guide them through the process of choosing a credential to present. That said, the interface to the chooser is as follows:

<section algorithm="ask the user to choose">
The user agent can <dfn export abstract-op local-lt="ask to choose">ask the user to choose a `Credential`</dfn>, given a {{CredentialRequestOptions}} |options|, and a set of {{Credential}} objects from the [=credential store=] |locally discovered credentials|.

This algorithm returns either `null` if the user chose not to share a credential with the site, a {{Credential}} object if the user chose a specific credential, or a {{Credential}} [=interface object=] if the user chose a type of credential.

It seems reasonable for the chooser interface to display the list of |locally discovered credentials| to the user, perhaps something like this exceptionally non-normative mock:

If the |options| provided is not <a>matchable a priori, then it might also make sense for the chooser interface to list the [=relevant credential interface objects=] for |options| that aren't covered by the list of explicit credentials. If, for instance, a site accepts webauthn-style authenticators, then "Security Key" might show up in the chooser list with an appropriate icon.

Also, note that in some cases the user agent may skip the chooser entirely. For example, if the only [=relevant credential interface objects=] is one that itself requires user interaction, the user agent may return that interface directly, and rely on its internal mediation flow for user consent.

</section>
</section>

<!--

-->

<section>
Security and Privacy Considerations # {#security-and-privacy}

The following sections represent guidelines for various security and privacy considerations. Individual credential types may enforce stricter or more relaxed versions of these guidelines.

Cross-domain credential access ## {#security-credential-access}

Credentials are sensitive information, and user agents need to exercise caution in determining when they can be safely shared with a website. The safest option is to restrict credential sharing to the exact origin on which they were saved. That is likely too restrictive for the web, however: consider sites which divide functionality into subdomains like `example.com` vs `admin.example.com`.

As a compromise between annoying users, and securing their credentials, user agents:

1. MUST NOT share credentials between origins whose scheme components represent a downgrade in security. That is, it may make sense to allow credentials saved on `http://example.com/` to be made available to `https://example.com/` (in order to encourage developers to migrate to secure transport), but the inverse would be dangerous.

2. MAY use the Public Suffix List [[!PSL]] to determine the effective scope of a credential by comparing the [=registerable domain=] of the credential's {{Credential/[[origin]]}} with the origin in which {{CredentialsContainer/get()}} is called. That is: credentials saved on

`https://admin.example.com/` and `https://example.com/` MAY be offered to users when {{CredentialsContainer/get()}} is called from `https://www.example.com/`, and vice versa.

3. MUST NOT offer credentials to an origin in response to {{CredentialsContainer/get()}} without [=user mediation=] if the credential's origin is not an exact match for the calling origin. That is, {{Credential}} objects for `https://example.com` would not be returned directly to `https://www.example.com`, but could be offered to the user via the chooser.

Credential Leakage ## {#security-leakage}

Developers are well-advised to take
some precautions to mitigate the risk that a cross-site scripting attack could turn into
persistent access to a user's account by setting a reasonable Content Security Policy [[!CSP]]
which restricts the endpoints to which data can be sent. In particular, developers should ensure
that the following directives are set, explicitly or implicitly, in their pages' policies:

* <a>`script-src` and <a>`object-src` both restrict script execution on a page, making it less likely that a cross-site scripting attack will succeed in the first place. If sites are populating <{form}> elements, also <a>`form-action` directives should be set.

* <a>`connect-src` restricts the origins to which {{fetch()}} may submit data (which mitigates the risk that credentials could be exfiltrated to `evil.com`.

* <a>`child-src` restricts the nested browsing contexts which may be embedded in a page, making it more difficult to inject a malicious `postMessage()` target. [[WEBMESSAGING]]

Developers should, of course, also properly escape input and output, and consider using other
layers of defense, such as Subresource Integrity [[SRI]] to further reduce risk.

When defining specific credential types, specific credential types SHOULD give due consideration
to the ways in which credential data can be transmitted over the wire. It might be reasonable,
for example, to define transmission mechanisms which are restricted to same-origin endpoints.

Insecure Sites ## {#insecure-sites}

User agents MUST NOT expose the APIs defined here to environments which are not [=secure contexts=]. User agents might implement autofill mechanisms which store user credentials and fill sign-in forms on non-<a>a priori authenticated URLs, but those sites cannot be trusted to interact directly with the credential manager in any meaningful way, and those sites MUST NOT have access to credentials saved in [=secure contexts=].

Origin Confusion ## {#security-origin-confusion}

If framed pages have access to the APIs defined here, it might be possible to confuse a user into granting access to credentials for an origin other than the <a>top-level browsing context, which is the only security origin which users can reasonably be expected to understand.

Therefore, both {{get()}} and {{store()}} will result in a rejected {{Promise}} when called from a [=nested browsing context=], and the API and related interfaces are not exposed inside {{Worker}} contexts.

The algorithms in [[#algorithm-request]] and [[#algorithm-store]] contain more detail.

Timing Attacks ## {#security-timing}

If the user has no credentials for an origin, a call to {{CredentialsContainer/get()}} will resolve very quickly indeed. A malicious website could distinguish between a user with no credentials and a user with credentials who chooses not to share them.

User agents SHOULD also rate-limit credential requests. It's almost certainly abusive for a page to request credentials more than a few times in a short period.

Signing-Out ## {#security-signout}

If a user has chosen to automatically sign-in to websites, as discussed in [[#user-mediation-requirement]], then the user agent will provide credentials to an origin whenever it asks for them. The website can instruct the user agent to suppress this behavior by calling {{CredentialsContainer}}'s {{CredentialsContainer/preventSilentAccess()}} method, which will turn off automatic sign-in for a given origin.

The user agent relies on the website to do the right thing; an inattentive (or malicious) website could simply neglect to call this method, causing the user agent to continue providing credentials against the user's apparent intention. This is marginally worse than the status-quo of a site that doesn't clear user credentials when they click "Sign-out", as the user agent becomes complicit in the authentication.

The user MUST have some control over this behavior. As noted in [[#user-mediation-requirement]], clearing cookies for an origin will also reset that origin's <a>`prevent silent access` flag the [=credential store=] to `true`. Additionally, the user agent SHOULD provide some UI affordance for disabling automatic sign-in for a particular origin. This could be tied to the notification that credentials have been provided to an origin, for example.

Chooser Leakage ## {#security-chooser-leakage}

If a user agent's [=credential chooser=] displays images supplied by an origin (for example, if a {{Credential}} displays a site's favicon), then, requests for these images MUST NOT be directly tied to instantiating the chooser in order to avoid leaking chooser usage. One option would be to fetch the images in the background when saving or updating a {{Credential}}, and to cache them for the lifetime of the {{Credential}}.

These images MUST be fetched with the [=request/credentials mode=] set to "`omit`", the [=request/service-workers mode=] set to "`none`", the [=request/client=] set to `null`, the [=request/initiator=] set to the empty string, and the [=request/destination=] "`subresource`".

Moreover, if the user agent allows the user to change either the name or icon associated with the credential, the alterations to the data SHOULD NOT be exposed to the website (consider a user who names two credentials for an origin "My fake account" and "My real account", for instance).

Locally Stored Data ## {#security-local-data}

This API offers an [=origin=] the ability to store data persistently along with a user's profile. Since most user agents treat credential data differently than "browsing data" (cookies, etc.) this might have the side effect of surprising a user who might believe that all traces of an origin have been wiped out when they clear their cookies.

User agents SHOULD provide UI that makes it clear to a user that credential data is stored for an origin, and SHOULD make it easy for users to remove such data when they're no longer interested in keeping it around.
</section>

<!--

-->
<section>
Implementation Considerations # {#implementation}

This section is non-normative.

Website Authors ## {#implementation-authors}

ISSUE(w3c/webappsec#290): Add some thoughts here about when and how the API should be used, especially with regard to {{CredentialRequestOptions/mediation}}.

ISSUE: Describe encoding restrictions of submitting credentials by {{fetch()}} with a {{FormData}} body.

The user agent relies on the website to do the right thing; an inattentive (or malicious) website could simply neglect to call this method, causing the user agent to continue providing credentials against the user's apparent intention. This is marginally worse than the status-quo of a site that doesn't clear user credentials when they click "Sign-out", as the user agent becomes complicit in the authentication.

The user MUST have some control over this behavior. As noted in [[#user-mediation-requirement]], clearing cookies for an origin will also reset that origin's <a>`prevent silent access` flag the [=credential store=] to `true`. Additionally, the user agent SHOULD provide some UI affordance for disabling automatic sign-in for a particular origin. This could be tied to the notification that credentials have been provided to an origin, for example.

Chooser Leakage ## {#security-chooser-leakage}

If a user agent's [=credential chooser=] displays images supplied by an origin (for example, if a {{Credential}} displays a site's favicon), then, requests for these images MUST NOT be directly tied to instantiating the chooser in order to avoid leaking chooser usage. One option would be to fetch the images in the background when saving or updating a {{Credential}}, and to cache them for the lifetime of the {{Credential}}.

These images MUST be fetched with the [=request/credentials mode=] set to "`omit`", the [=request/service-workers mode=] set to "`none`", the [=request/client=] set to `null`, the [=request/initiator=] set to the empty string, and the [=request/destination=] "`subresource`".

Moreover, if the user agent allows the user to change either the name or icon associated with the credential, the alterations to the data SHOULD NOT be exposed to the website (consider a user who names two credentials for an origin "My fake account" and "My real account", for instance).

Locally Stored Data ## {#security-local-data}

This API offers an [=origin=] the ability to store data persistently along with a user's profile. Since most user agents treat credential data differently than "browsing data" (cookies, etc.) this might have the side effect of surprising a user who might believe that all traces of an origin have been wiped out when they clear their cookies.

User agents SHOULD provide UI that makes it clear to a user that credential data is stored for an origin, and SHOULD make it easy for users to remove such data when they're no longer interested in keeping it around.
</section>

<!--

-->
<section>
Implementation Considerations # {#implementation}

This section is non-normative.

Website Authors ## {#implementation-authors}

ISSUE(w3c/webappsec#290): Add some thoughts here about when and how the API should be used, especially with regard to {{CredentialRequestOptions/mediation}}.

ISSUE: Describe encoding restrictions of submitting credentials by {{fetch()}} with a {{FormData}} body.

Extension Points ## {#implementation-extension}

This document provides a generic, high-level API that's meant to be extended with specific types of [=credentials=] that serve specific authentication needs. Doing so is, hopefully, straightforward:

1. Define a new interface that inherits from {{Credential}}:

     <pre>
       [Exposed=Window,
        SecureContext]
       interface ExampleCredential : Credential {
         // Definition goes here.
       };
     </pre>

2. Define appropriate {{Credential/[[Create]](options)}}, {{Credential/[[CollectFromCredentialStore]](origin, options)}}, {{Credential/[[DiscoverFromExternalSource]](origin, options)}}, and {{Credential/[[Store]](credential)}} methods on `ExampleCredential`'s [=interface object=]. {{Credential/[[CollectFromCredentialStore]](origin, options)}} is appropriate for [=credentials=] that remain [=effective=] forever and can therefore simply be copied out of the [=credential store=], while {{Credential/[[DiscoverFromExternalSource]](origin, options)}} is appropriate for [=credentials=] that need to be re-generated from a [=credential source=].

     `ExampleCredential`'s `[[CollectFromCredentialStore]](origin, options)` internal method is called with an [=environment settings object/origin=] (`origin`) and a CredentialRequestOptions object (`options`), and returns a set of {{Credential}} objects that match the options provided. If no matching {{Credential}} objects are available, the returned set will be empty.

     1. Assert: `options`[`example`] exists.

     2. If `options`[`example`] is not truthy, return the empty set.

     3. For each credential in the [=credential store=]:

        1. ...

3. Define the value of the `ExampleCredential` [=interface object=]'s {{[[type]]}} slot:

     The `ExampleCredential` [=interface object=] has an internal slot named `[[type]]` whose value is the string "`example`".

4. Define the value of the `ExampleCredential` [=interface object=]'s {{[[discovery]]}} slot:

     The `ExampleCredential` [=interface object=] has an internal slot named `[[type]]` whose value is "{{Credential/[[discovery]]/credential store}}".

4. Extend {{CredentialRequestOptions}} with the options the new credential type needs to respond reasonably to {{get()}}:

     <pre>
       dictionary ExampleCredentialRequestOptions {
         // Definition goes here.
       };

       partial dictionary CredentialRequestOptions {

```
            ExampleCredentialRequestOptions example;
          };
        </pre>
      </div>

      5. Extend {{CredentialCreationOptions}} with the data the new credential type needs to create
         `Credential` objects in response to {{create()}}:

      <div class="example">
        <pre>
          dictionary ExampleCredentialInit {
            // Definition goes here.
          };

          partial dictionary CredentialCreationOptions {
            ExampleCredentialInit example;
          };
        </pre>
      </div>

    You might also find that new primitives are necessary. For instance, you might want to return
    many {{Credential}} objects rather than just one in some sort of complicated, multi-factor
    sign-in process. That might be accomplished in a generic fashion by adding a `getAll()` method to
    {{CredentialsContainer}} which returned a `sequence<Credential>`, and defining a reasonable
    mechanism for dealing with requesting credentials of distinct types.

    For any such extension, we recommend getting in touch with
    <a href="mailto:public-webappsec@w3.org">public-webappsec@</a> for consultation and review.

    ## Browser Extensions ## {#browser-extensions}

    Ideally, user agents that implement an extension system of some sort will
    allow third-parties to hook into these API endpoints in order to improve
    the behavior of third party credential management software in the same way
    that user agents can improve their own via this imperative approach.

    This could range from a complex new API that the user agent mediates, or
    simply by allowing extensions to overwrite the {{CredentialsContainer/get()}}
    and {{CredentialsContainer/store()}} endpoints for their own purposes.
  </section>

  <!--
  [redacted]
  -->
  <section>
    # Future Work # {#teh-futur}

    <em>This section is non-normative.</em>

    The API defined here does the bare minimum to expose user agent's credential managers to the web,
    and allows the web to help those credential managers understand when federated identity providers
    are in use. The next logical step will be along the lines sketched in documents like [[WEB-LOGIN]]
    (and, to some extent, Mozilla's BrowserID [[BROWSERID]]).

    The user agent is in the unique position of being able to effectively mediate the relationship
    between users, identity providers, and websites. If the user agent can remove some of the risk and
    confusion associated with the typical authentication flows, users will be in a significantly
    better position than today.

    A natural way to expose this information might be to extend the {{FederatedCredential}} interface
```

/Users/jehodges/Documents/work/standards/W3C/webappsec/equalsjeffh/webappsec-credential-management/index-issue-92-accessing-settings-object-b34d4c8.src.html, Top line: 2016

with properties like authentication tokens, and possibly to add some form of manifest format with properties that declare the authentication type which the provider supports.

The API described here is designed to be extensible enough to support use cases that require user interaction, perhaps with websites other than the one which requested credentials. We hope that the Promise-based system we've settled on is extensible enough to support these kinds of asynchronous flows which could require some level of interaction between multiple browsing contexts (e.g. mediated activity on <code>idp.com</code> might resolve a Promise handed back to <code>rp.com</code>) or between devices and user agents (e.g. [[WEBAUTHN]]) in the future without redesigning the API from the ground up.

Baby steps.
</section>

/Users/jehodges/Documents/work/standards/W3C/webappsec/equalsjeffh/webappsec-credential-management/index.src.html, Top line: 2070

with properties like authentication tokens, and possibly to add some form of manifest format with properties that declare the authentication type which the provider supports.

The API described here is designed to be extensible enough to support use cases that require user interaction, perhaps with websites other than the one which requested credentials. We hope that the Promise-based system we've settled on is extensible enough to support these kinds of asynchronous flows which could require some level of interaction between multiple browsing contexts (e.g. mediated activity on <code>idp.com</code> might resolve a Promise handed back to <code>rp.com</code>) or between devices and user agents (e.g. [[WEBAUTHN]]) in the future without redesigning the API from the ground up.

Baby steps.
</section>