

```
0001 <h1>Credential Management Level 1</h1>
0002 <pre class="metadata">
0003 Status: ED
0004 ED: https://w3c.github.io/webappsec-credential-management/
0005 TR: http://www.w3.org/TR/credential-management-1/
0006 Previous Version: http://www.w3.org/TR/2015/WD-credential-management-1-20150430/
0007 Shortname: credential-management
0008 Level: 1
0009 Editor: Mike West 56384, Google Inc., mkwst@google.com
0010 Group: webappsec
0011 Abstract:
0012 This specification describes an imperative API enabling a website to request a
0013 user's credentials from a user agent, and to help the user agent correctly
0014 store user credentials for future use.
0015 Indent: 2
0016 Version History: https://github.com/w3c/webappsec-credential-
0017 management/commits/master/index.src.html
0018 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>)
0019 Markup Shorthands: css off, markdown on
0020 </pre>
0021 <pre class="anchors">
0022 spec: ECMA262; urlPrefix: https://tc39.github.io/ecma262/
0023 type: dfn
0024 text: JavaScript realm; url: sec-code-realms
0025 spec: HTML; urlPrefix: https://html.spec.whatwg.org/multipage/
0026 urlPrefix: forms.html
0027 type: element-attr
0028 text: autocomplete; for: input; url: #attr-fe-autocomplete
0029 text: name; for: input; url: #attr-fe-name
0030 text: enctype; for: form; url: #concept-fs-enctype
0031 type: attr-value
0032 for: autocomplete
0033 text: current-password; url: attr-fe-autocomplete-current-password
0034 text: new-password; url: attr-fe-autocomplete-new-password
0035 text: nickname; url: attr-fe-autocomplete-nickname
0036 text: name; url: attr-fe-autocomplete-name
0037 text: photo; url: attr-fe-autocomplete-photo
0038 text: username; url: attr-fe-autocomplete-username
0039 spec: XHR; urlPrefix: https://xhr.spec.whatwg.org/
0040 type: dfn
0041 text: entry; url: concept-formdata-entry
0042 text: entries; for: FormData; url: concept-formdata-entry
0043 text: name; for: entry; url: concept-formdata-entry-name
0044 text: value; for: entry; url: concept-formdata-entry-value
0045 text: type; for: entry; url: concept-formdata-entry-type
0046 type: interface
0047 text: FormData; url: interface-formdata
0048 spec: PSL; urlPrefix: https://publicsuffix.org/list/
0049 type: dfn
0050 text: registerable domain; url: #
0051 text: public suffix; url: #
0052 spec: FETCH; urlPrefix: https://fetch.spec.whatwg.org/
0053 type: dfn
0054 text: http-network-or-cache fetch; url: http-network-or-cache-fetch
0055 </pre>
0056 <pre class="link-defaults">
0057 spec:html; type:dfn; for:/:; text:origin
0058 spec:fetch; type:dfn; for:/:; text:request
0059 spec:fetch; type:dictionary; for:/:; text:RequestInit
0060
0061 spec:infra; type:dfn; for:/:; text:set
0062 spec:infra; type:dfn; for:struct; text:item
0063 spec:webidl; type:idl; for:/:; text:Function
0064
0065 <!-- These need to be exported -->
0066 spec:html; type:dfn; text:submittable element
0067 spec:html; type:dfn; text:form owner
```

```
0001 <h1>Credential Management Level 1</h1>
0002 <pre class="metadata">
0003 Status: ED
0004 ED: https://w3c.github.io/webappsec-credential-management/
0005 TR: http://www.w3.org/TR/credential-management-1/
0006 Previous Version: http://www.w3.org/TR/2015/WD-credential-management-1-20150430/
0007 Shortname: credential-management
0008 Level: 1
0009 Editor: Mike West 56384, Google Inc., mkwst@google.com
0010 Group: webappsec
0011 Abstract:
0012 This specification describes an imperative API enabling a website to request a
0013 user's credentials from a user agent, and to help the user agent correctly
0014 store user credentials for future use.
0015 Indent: 2
0016 Version History: https://github.com/w3c/webappsec-credential-
0017 management/commits/master/index.src.html
0018 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>)
0019 Markup Shorthands: css off, markdown on
0020 </pre>
0021 <pre class="anchors">
0022 spec: ECMA262; urlPrefix: https://tc39.github.io/ecma262/
0023 type: dfn
0024 text: JavaScript realm; url: sec-code-realms
0025 spec: HTML; urlPrefix: https://html.spec.whatwg.org/multipage/
0026 urlPrefix: forms.html
0027 type: element-attr
0028 text: autocomplete; for: input; url: #attr-fe-autocomplete
0029 text: name; for: input; url: #attr-fe-name
0030 text: enctype; for: form; url: #concept-fs-enctype
0031 type: attr-value
0032 for: autocomplete
0033 text: current-password; url: attr-fe-autocomplete-current-password
0034 text: new-password; url: attr-fe-autocomplete-new-password
0035 text: nickname; url: attr-fe-autocomplete-nickname
0036 text: name; url: attr-fe-autocomplete-name
0037 text: photo; url: attr-fe-autocomplete-photo
0038 text: username; url: attr-fe-autocomplete-username
0039 spec: XHR; urlPrefix: https://xhr.spec.whatwg.org/
0040 type: dfn
0041 text: entry; url: concept-formdata-entry
0042 text: entries; for: FormData; url: concept-formdata-entry
0043 text: name; for: entry; url: concept-formdata-entry-name
0044 text: value; for: entry; url: concept-formdata-entry-value
0045 text: type; for: entry; url: concept-formdata-entry-type
0046 type: interface
0047 text: FormData; url: interface-formdata
0048 spec: PSL; urlPrefix: https://publicsuffix.org/list/
0049 type: dfn
0050 text: registerable domain; url: #
0051 text: public suffix; url: #
0052 spec: FETCH; urlPrefix: https://fetch.spec.whatwg.org/
0053 type: dfn
0054 text: http-network-or-cache fetch; url: http-network-or-cache-fetch
0055 </pre>
0056 <pre class="link-defaults">
0057 spec:html; type:dfn; for:/:; text:origin
0058 spec:fetch; type:dfn; for:/:; text:request
0059 spec:fetch; type:dictionary; for:/:; text:RequestInit
0060 spec:infra; type:dfn; for:/:; text:set
0061 spec:infra; type:dfn; for:struct; text:item
0062 spec:webidl; type:idl; for:/:; text:Function
0063
0064 <!-- These need to be exported -->
0065 spec:html; type:dfn; text:submittable element
0066 spec:html; type:dfn; text:form owner
```

```

0064 spec:html; type:dfn; text:autofill detail tokens
0065 spec:url; type:dfn; text:urlencoded byte serializer
0066 </pre>

<pre class="biblio">
0067 {
0068   "WEB-LOGIN": {
0069     "authors": [ "Jason Denizac", "Robin Berjon", "Anne van Kesteren" ],
0070     "href": "https://github.com/jden/web-login",
0071     "title": "web-login"
0072   }
0073 }
0074 </pre>

<!--
0075 INTRO
0076 DUCTION
0077 SECTION
0078 START
0079 HERE
0080 WITH
0081 A
0082 SECTION
0083 OF
0084 INTRODUCTION
0085 TEXT
0086 AND
0087 A
0088 SECTION
0089 OF
0090 INTRODUCTION
0091 TEXT
0092 AND
0093 A
0094 SECTION
0095 OF
0096 INTRODUCTION
0097 TEXT
0098 AND
0099 A
0100 SECTION
0101 OF
0102 INTRODUCTION
0103 TEXT
0104 AND
0105 A
0106 SECTION
0107 OF
0108 INTRODUCTION
0109 TEXT
0110 AND
0111 A
0112 SECTION
0113 OF
0114 INTRODUCTION
0115 TEXT
0116 AND
0117 A
0118 SECTION
0119 OF
0120 INTRODUCTION
0121 TEXT
0122 AND
0123 A
0124 SECTION
0125 OF
0126 INTRODUCTION
0127 TEXT
0128 AND
0129 A
0130 SECTION
0131 OF
0132 INTRODUCTION
0133 TEXT
0134 AND
0135 A
0136 SECTION
0137 OF
0138 INTRODUCTION
0139 TEXT
0140 AND
0141 A
0142 SECTION
0143 OF
0144 INTRODUCTION
0145 TEXT
0146 AND
0147 A
0148 SECTION
0149 OF
0150 INTRODUCTION
0151 TEXT
0152 AND
0153 A
0154 SECTION
0155 OF
0156 INTRODUCTION
0157 TEXT
0158 AND
0159 A
0160 SECTION
0161 OF
0162 INTRODUCTION
0163 TEXT
0164 AND
0165 A
0166 SECTION
0167 OF
0168 INTRODUCTION
0169 TEXT
0170 AND
0171 A
0172 SECTION
0173 OF
0174 INTRODUCTION
0175 TEXT
0176 AND
0177 A
0178 SECTION
0179 OF
0180 INTRODUCTION
0181 TEXT
0182 AND
0183 A
0184 SECTION
0185 OF
0186 INTRODUCTION
0187 TEXT
0188 AND
0189 A
0190 SECTION
0191 OF
0192 INTRODUCTION
0193 TEXT
0194 AND
0195 A
0196 SECTION
0197 OF
0198 INTRODUCTION
0199 TEXT
0200 AND
0201 A
0202 SECTION
0203 OF
0204 INTRODUCTION
0205 TEXT
0206 AND
0207 A
0208 SECTION
0209 OF
0210 INTRODUCTION
0211 TEXT
0212 AND
0213 A
0214 SECTION
0215 OF
0216 INTRODUCTION
0217 TEXT
0218 AND
0219 A
0220 SECTION
0221 OF
0222 INTRODUCTION
0223 TEXT
0224 AND
0225 A
0226 SECTION
0227 OF
0228 INTRODUCTION
0229 TEXT
0230 AND
0231 A
0232 SECTION
0233 OF
0234 INTRODUCTION
0235 TEXT
0236 AND
0237 A
0238 SECTION
0239 OF
0240 INTRODUCTION
0241 TEXT
0242 AND
0243 A
0244 SECTION
0245 OF
0246 INTRODUCTION
0247 TEXT
0248 AND
0249 A
0250 SECTION
0251 OF
0252 INTRODUCTION
0253 TEXT
0254 AND
0255 A
0256 SECTION
0257 OF
0258 INTRODUCTION
0259 TEXT
0260 AND
0261 A
0262 SECTION
0263 OF
0264 INTRODUCTION
0265 TEXT
0266 AND
0267 A
0268 SECTION
0269 OF
0270 INTRODUCTION
0271 TEXT
0272 AND
0273 A
0274 SECTION
0275 OF
0276 INTRODUCTION
0277 TEXT
0278 AND
0279 A
0280 SECTION
0281 OF
0282 INTRODUCTION
0283 TEXT
0284 AND
0285 A
0286 SECTION
0287 OF
0288 INTRODUCTION
0289 TEXT
0290 AND
0291 A
0292 SECTION
0293 OF
0294 INTRODUCTION
0295 TEXT
0296 AND
0297 A
0298 SECTION
0299 OF
0300 INTRODUCTION
0301 TEXT
0302 AND
0303 A
0304 SECTION
0305 OF
0306 INTRODUCTION
0307 TEXT
0308 AND
0309 A
0310 SECTION
0311 OF
0312 INTRODUCTION
0313 TEXT
0314 AND
0315 A
0316 SECTION
0317 OF
0318 INTRODUCTION
0319 TEXT
0320 AND
0321 A
0322 SECTION
0323 OF
0324 INTRODUCTION
0325 TEXT
0326 AND
0327 A
0328 SECTION
0329 OF
0330 INTRODUCTION
0331 TEXT
0332 AND
0333 A
0334 SECTION
0335 OF
0336 INTRODUCTION
0337 TEXT
0338 AND
0339 A
0340 SECTION
0341 OF
0342 INTRODUCTION
0343 TEXT
0344 AND
0345 A
0346 SECTION
0347 OF
0348 INTRODUCTION
0349 TEXT
0350 AND
0351 A
0352 SECTION
0353 OF
0354 INTRODUCTION
0355 TEXT
0356 AND
0357 A
0358 SECTION
0359 OF
0360 INTRODUCTION
0361 TEXT
0362 AND
0363 A
0364 SECTION
0365 OF
0366 INTRODUCTION
0367 TEXT
0368 AND
0369 A
0370 SECTION
0371 OF
0372 INTRODUCTION
0373 TEXT
0374 AND
0375 A
0376 SECTION
0377 OF
0378 INTRODUCTION
0379 TEXT
0380 AND
0381 A
0382 SECTION
0383 OF
0384 INTRODUCTION
0385 TEXT
0386 AND
0387 A
0388 SECTION
0389 OF
0390 INTRODUCTION
0391 TEXT
0392 AND
0393 A
0394 SECTION
0395 OF
0396 INTRODUCTION
0397 TEXT
0398 AND
0399 A
0400 SECTION
0401 OF
0402 INTRODUCTION
0403 TEXT
0404 AND
0405 A
0406 SECTION
0407 OF
0408 INTRODUCTION
0409 TEXT
0410 AND
0411 A
0412 SECTION
0413 OF
0414 INTRODUCTION
0415 TEXT
0416 AND
0417 A
0418 SECTION
0419 OF
0420 INTRODUCTION
0421 TEXT
0422 AND
0423 A
0424 SECTION
0425 OF
0426 INTRODUCTION
0427 TEXT
0428 AND
0429 A
0430 SECTION
0431 OF
0432 INTRODUCTION
0433 TEXT
0434 AND
0435 A
0436 SECTION
0437 OF
0438 INTRODUCTION
0439 TEXT
0440 AND
0441 A
0442 SECTION
0443 OF
0444 INTRODUCTION
0445 TEXT
0446 AND
0447 A
0448 SECTION
0449 OF
0450 INTRODUCTION
0451 TEXT
0452 AND
0453 A
0454 SECTION
0455 OF
0456 INTRODUCTION
0457 TEXT
0458 AND
0459 A
0460 SECTION
0461 OF
0462 INTRODUCTION
0463 TEXT
0464 AND
0465 A
0466 SECTION
0467 OF
0468 INTRODUCTION
0469 TEXT
0470 AND
0471 A
0472 SECTION
0473 OF
0474 INTRODUCTION
0475 TEXT
0476 AND
0477 A
0478 SECTION
0479 OF
0480 INTRODUCTION
0481 TEXT
0482 AND
0483 A
0484 SECTION
0485 OF
0486 INTRODUCTION
0487 TEXT
0488 AND
0489 A
0490 SECTION
0491 OF
0492 INTRODUCTION
0493 TEXT
0494 AND
0495 A
0496 SECTION
0497 OF
0498 INTRODUCTION
0499 TEXT
0500 AND
0501 A
0502 SECTION
0503 OF
0504 INTRODUCTION
0505 TEXT
0506 AND
0507 A
0508 SECTION
0509 OF
0510 INTRODUCTION
0511 TEXT
0512 AND
0513 A
0514 SECTION
0515 OF
0516 INTRODUCTION
0517 TEXT
0518 AND
0519 A
0520 SECTION
0521 OF
0522 INTRODUCTION
0523 TEXT
0524 AND
0525 A
0526 SECTION
0527 OF
0528 INTRODUCTION
0529 TEXT
0530 AND
0531 A
0532 SECTION
0533 OF
0534 INTRODUCTION
0535 TEXT
0536 AND
0537 A
0538 SECTION
0539 OF
0540 INTRODUCTION
0541 TEXT
0542 AND
0543 A
0544 SECTION
0545 OF
0546 INTRODUCTION
0547 TEXT
0548 AND
0549 A
0550 SECTION
0551 OF
0552 INTRODUCTION
0553 TEXT
0554 AND
0555 A
0556 SECTION
0557 OF
0558 INTRODUCTION
0559 TEXT
0560 AND
0561 A
0562 SECTION
0563 OF
0564 INTRODUCTION
0565 TEXT
0566 AND
0567 A
0568 SECTION
0569 OF
0570 INTRODUCTION
0571 TEXT
0572 AND
0573 A
0574 SECTION
0575 OF
0576 INTRODUCTION
0577 TEXT
0578 AND
0579 A
0580 SECTION
0581 OF
0582 INTRODUCTION
0583 TEXT
0584 AND
0585 A
0586 SECTION
0587 OF
0588 INTRODUCTION
0589 TEXT
0590 AND
0591 A
0592 SECTION
0593 OF
0594 INTRODUCTION
0595 TEXT
0596 AND
0597 A
0598 SECTION
0599 OF
0600 INTRODUCTION
0601 TEXT
0602 AND
0603 A
0604 SECTION
0605 OF
0606 INTRODUCTION
0607 TEXT
0608 AND
0609 A
0610 SECTION
0611 OF
0612 INTRODUCTION
0613 TEXT
0614 AND
0615 A
0616 SECTION
0617 OF
0618 INTRODUCTION
0619 TEXT
0620 AND
0621 A
0622 SECTION
0623 OF
0624 INTRODUCTION
0625 TEXT
0626 AND
0627 A
0628 SECTION
0629 OF
0630 INTRODUCTION
0631 TEXT
0632 AND
0633 A
0634 SECTION
0635 OF
0636 INTRODUCTION
0637 TEXT
0638 AND
0639 A
0640 SECTION
0641 OF
0642 INTRODUCTION
0643 TEXT
0644 AND
0645 A
0646 SECTION
0647 OF
0648 INTRODUCTION
0649 TEXT
0650 AND
0651 A
0652 SECTION
0653 OF
0654 INTRODUCTION
0655 TEXT
0656 AND
0657 A
0658 SECTION
0659 OF
0660 INTRODUCTION
0661 TEXT
0662 AND
0663 A
0664 SECTION
0665 OF
0666 INTRODUCTION
0667 TEXT
0668 AND
0669 A
0670 SECTION
0671 OF
0672 INTRODUCTION
0673 TEXT
0674 AND
0675 A
0676 SECTION
0677 OF
0678 INTRODUCTION
0679 TEXT
0680 AND
0681 A
0682 SECTION
0683 OF
0684 INTRODUCTION
0685 TEXT
0686 AND
0687 A
0688 SECTION
0689 OF
0690 INTRODUCTION
0691 TEXT
0692 AND
0693 A
0694 SECTION
0695 OF
0696 INTRODUCTION
0697 TEXT
0698 AND
0699 A
0700 SECTION
0701 OF
0702 INTRODUCTION
0703 TEXT
0704 AND
0705 A
0706 SECTION
0707 OF
0708 INTRODUCTION
0709 TEXT
0710 AND
0711 A
0712 SECTION
0713 OF
0714 INTRODUCTION
0715 TEXT
0716 AND
0717 A
0718 SECTION
0719 OF
0720 INTRODUCTION
0721 TEXT
0722 AND
0723 A
0724 SECTION
0725 OF
0726 INTRODUCTION
0727 TEXT
0728 AND
0729 A
0730 SECTION
0731 OF
0732 INTRODUCTION
0733 TEXT
0734 AND
0735 A
0736 SECTION
0737 OF
0738 INTRODUCTION
0739 TEXT
0740 AND
0741 A
0742 SECTION
0743 OF
0744 INTRODUCTION
0745 TEXT
0746 AND
0747 A
0748 SECTION
0749 OF
0750 INTRODUCTION
0751 TEXT
0752 AND
0753 A
0754 SECTION
0755 OF
0756 INTRODUCTION
0757 TEXT
0758 AND
0759 A
0760 SECTION
0761 OF
0762 INTRODUCTION
0763 TEXT
0764 AND
0765 A
0766 SECTION
0767 OF
0768 INTRODUCTION
0769 TEXT
0770 AND
0771 A
0772 SECTION
0773 OF
0774 INTRODUCTION
0775 TEXT
0776 AND
0777 A
0778 SECTION
0779 OF
0780 INTRODUCTION
0781 TEXT
0782 AND
0783 A
0784 SECTION
0785 OF
0786 INTRODUCTION
0787 TEXT
0788 AND
0789 A
0790 SECTION
0791 OF
0792 INTRODUCTION
0793 TEXT
0794 AND
0795 A
0796 SECTION
0797 OF
0798 INTRODUCTION
0799 TEXT
0800 AND
0801 A
0802 SECTION
0803 OF
0804 INTRODUCTION
0805 TEXT
0806 AND
0807 A
0808 SECTION
0809 OF
0810 INTRODUCTION
0811 TEXT
0812 AND
0813 A
0814 SECTION
0815 OF
0816 INTRODUCTION
0817 TEXT
0818 AND
0819 A
0820 SECTION
0821 OF
0822 INTRODUCTION
0823 TEXT
0824 AND
0825 A
0826 SECTION
0827 OF
0828 INTRODUCTION
0829 TEXT
0830 AND
0831 A
0832 SECTION
0833 OF
0834 INTRODUCTION
0835 TEXT
0836 AND
0837 A
0838 SECTION
0839 OF
0840 INTRODUCTION
0841 TEXT
0842 AND
0843 A
0844 SECTION
0845 OF
0846 INTRODUCTION
0847 TEXT
0848 AND
0849 A
0850 SECTION
0851 OF
0852 INTRODUCTION
0853 TEXT
0854 AND
0855 A
0856 SECTION
0857 OF
0858 INTRODUCTION
0859 TEXT
0860 AND
0861 A
0862 SECTION
0863 OF
0864 INTRODUCTION
0865 TEXT
0866 AND
0867 A
0868 SECTION
0869 OF
0870 INTRODUCTION
0871 TEXT
0872 AND
0873 A
0874 SECTION
0875 OF
0876 INTRODUCTION
0877 TEXT
0878 AND
0879 A
0880 SECTION
0881 OF
0882 INTRODUCTION
0883 TEXT
0884 AND
0885 A
0886 SECTION
0887 OF
0888 INTRODUCTION
0889 TEXT
0890 AND
0891 A
0892 SECTION
0893 OF
0894 INTRODUCTION
0895 TEXT
0896 AND
0897 A
0898 SECTION
0899 OF
0900 INTRODUCTION
0901 TEXT
0902 AND
0903 A
0904 SECTION
0905 OF
0906 INTRODUCTION
0907 TEXT
0908 AND
0909 A
0910 SECTION
0911 OF
0912 INTRODUCTION
0913 TEXT
0914 AND
0915 A
0916 SECTION
0917 OF
0918 INTRODUCTION
0919 TEXT
0920 AND
0921 A
0922 SECTION
0923 OF
0924 INTRODUCTION
0925 TEXT
0926 AND
0927 A
0928 SECTION
0929 OF
0930 INTRODUCTION
0931 TEXT
0932 AND
0933 A
0934 SECTION
0935 OF
0936 INTRODUCTION
0937 TEXT
0938 AND
0939 A
0940 SECTION
0941 OF
0942 INTRODUCTION
0943 TEXT
0944 AND
0945 A
0946 SECTION
0947 OF
0948 INTRODUCTION
0949 TEXT
0950 AND
0951 A
0952 SECTION
0953 OF
0954 INTRODUCTION
0955 TEXT
0956 AND
0957 A
0958 SECTION
0959 OF
0960 INTRODUCTION
0961 TEXT
0962 AND
0963 A
0964 SECTION
0965 OF
0966 INTRODUCTION
0967 TEXT
0968 AND
0969 A
0970 SECTION
0971 OF
0972 INTRODUCTION
0973 TEXT
0974 AND
0975 A
0976 SECTION
0977 OF
0978 INTRODUCTION
0979 TEXT
0980 AND
0981 A
0982 SECTION
0983 OF
0984 INTRODUCTION
0985 TEXT
0986 AND
0987 A
0988 SECTION
0989 OF
0990 INTRODUCTION
0991 TEXT
0992 AND
0993 A
0994 SECTION
0995 OF
0996 INTRODUCTION
0997 TEXT
0998 AND
0999 A
1000 SECTION
1001 OF
1002 INTRODUCTION
1003 TEXT
1004 AND
1005 A
1006 SECTION
1007 OF
1008 INTRODUCTION
1009 TEXT
1010 AND
1011 A
1012 SECTION
1013 OF
1014 INTRODUCTION
1015 TEXT
1016 AND
1017 A
1018 SECTION
1019 OF
1020 INTRODUCTION
1021 TEXT
1022 AND
1023 A
1024 SECTION
1025 OF
1026 INTRODUCTION
1027 TEXT
1028 AND
1029 A
1030 SECTION
1031 OF
1032 INTRODUCTION
1033 TEXT
1034 AND
1035 A
1036 SECTION
1037 OF
1038 INTRODUCTION
1039 TEXT
1040 AND
1041 A
1042 SECTION
1043 OF
1044 INTRODUCTION
1045 TEXT
1046 AND
1047 A
1048 SECTION
1049 OF
1050 INTRODUCTION
1051 TEXT
1052 AND
1053 A
1054 SECTION
1055 OF
1056 INTRODUCTION
1057 TEXT
1058 AND
1059 A
1060 SECTION
1061 OF
1062 INTRODUCTION
1063 TEXT
1064 AND
1065 A
1066 SECTION
1067 OF
1068 INTRODUCTION
1069 TEXT
1070 AND
1071 A
1072 SECTION
1073 OF
1074 INTRODUCTION
1075 TEXT
1076 AND
1077 A
1078 SECTION
1079 OF
1080 INTRODUCTION
1081 TEXT
1082 AND
1
```

0129 behavior is completely invisible: the website doesn't know that passwords have been stored, and
0130 it isn't notified that passwords have been filled. This is both good and bad. On the one hand, a
0131 user agent's password manager works regardless of whether or not a site cooperates, which is
0132 excellent for users. On the other, the password managers' behaviors are a fragile and proprietary
0133 hodgepodge of heuristics meant to detect and fill sign-in forms, password change forms, etc.
0134

0135 behavior is completely invisible: the website doesn't know that passwords have been stored, and
0136 it isn't notified that passwords have been filled. This is both good and bad. On the one hand, a
0137 user agent's password manager works regardless of whether or not a site cooperates, which is
0138 excellent for users. On the other, the password managers' behaviors are a fragile and proprietary
0139 hodgepodge of heuristics meant to detect and fill sign-in forms, password change forms, etc.
0140

0135 A few problems with the status quo stand out as being particularly noteworthy:

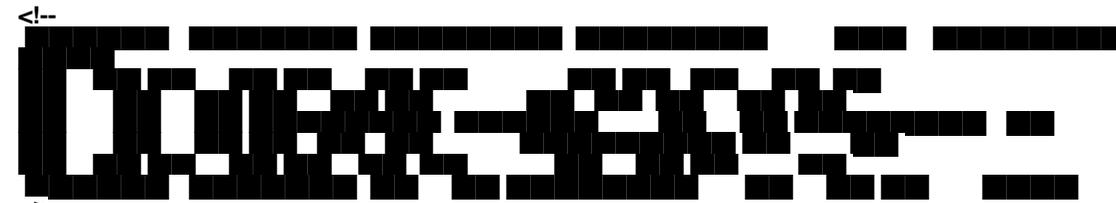
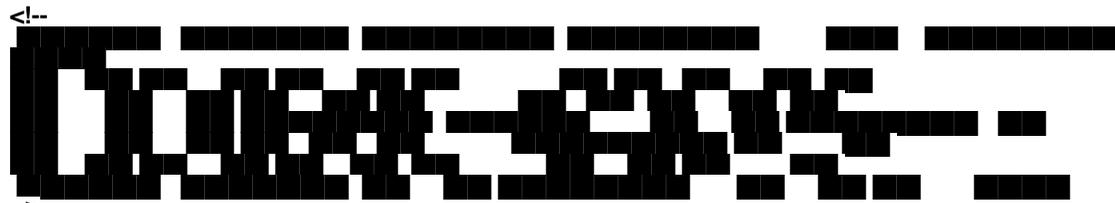
0141 A few problems with the status quo stand out as being particularly noteworthy:

- 0137 * User agents have an incredibly difficult time helping users with
0138 federated identity providers. While detecting a username/password
0139 form submission is fairly straightforward, detecting sign-in via a
0140 third-party is quite difficult to reliably do well. It would be nice
0141 if a website could help the user agent understand the intent behind the
0142 redirects associated with a typical federated sign-in action.
- 0143 * Likewise, user agents struggle to detect more esoteric sign-in
0144 mechanisms than simple username/password forms. Authors increasingly
0145 asynchronously sign users in via `{{XMLHttpRequest}}` or similar
0146 mechanisms in order to improve the experience and take more control over
0147 the presentation. This is good for users, but tough for user agents to
0148 integrate into their password managers. It would be nice if a website
0149 could help the user agent make sense of the sign-in mechanism they
0150 choose to use.
- 0151 * Finally, changing passwords is less well-supported than it could be if
0152 the website explicitly informed the user agent that credentials had
0153 changed.

- 0143 * User agents have an incredibly difficult time helping users with
0144 federated identity providers. While detecting a username/password
0145 form submission is fairly straightforward, detecting sign-in via a
0146 third-party is quite difficult to reliably do well. It would be nice
0147 if a website could help the user agent understand the intent behind the
0148 redirects associated with a typical federated sign-in action.
- 0149 * Likewise, user agents struggle to detect more esoteric sign-in
0150 mechanisms than simple username/password forms. Authors increasingly
0151 asynchronously sign users in via `{{XMLHttpRequest}}` or similar
0152 mechanisms in order to improve the experience and take more control over
0153 the presentation. This is good for users, but tough for user agents to
0154 integrate into their password managers. It would be nice if a website
0155 could help the user agent make sense of the sign-in mechanism they
0156 choose to use.
- 0157 * Finally, changing passwords is less well-supported than it could be if
0158 the website explicitly informed the user agent that credentials had
0159 changed.

0156 </section>

0162 </section>



0167 <section>
0168 # Core API # {#core}

0173 <section>
0174 # Core API # {#core}

0170 From a developer's perspective, a `<dfn export id="concept-credential">credential</dfn>` is an
0171 object which allows a developer to make an authentication decision for a particular action. This
0172 section defines a generic and extensible `{{Credential}}` interface which serves as a base class
0173 for `[=credentials=]` defined in this and other documents, along with a set of APIs
0174 hanging off of `{{CredentialsContainerNavigator.credentials.*}}` which enable developers to
0175 obtain them.

0176 From a developer's perspective, a `<dfn export id="concept-credential">credential</dfn>` is an
0177 object which allows a developer to make an authentication decision for a particular action. This
0178 section defines a generic and extensible `{{Credential}}` interface which serves as a base class
0179 for `[=credentials=]` defined in this and other documents, along with a set of APIs
0180 hanging off of `{{CredentialsContainerNavigator.credentials.*}}` which enable developers to
0181 obtain them.

0177 Various types of `[=credentials=]` are represented to JavaScript as an interface which
0178 `[=interface/inherits=]`, either directly or indirectly, from the `{{Credential}}` interface. This
0179 document defines two such interfaces, `{{PasswordCredential}}` and `{{FederatedCredential}}`.

0182 Various types of `[=credentials=]` are represented to JavaScript as an interface which
0183 `[=interface/inherits=]`, either directly or indirectly, from the `{{Credential}}` interface. This
0184 document defines two such interfaces, `{{PasswordCredential}}` and `{{FederatedCredential}}`.

0181 A `[=credential=]` is `<dfn export for="credential">effective</dfn>` for a particular `[=origin=]` if it
0182 is accepted as authentication on that origin. Even if a credential is effective at a particular
0183 point in time, the UA can't assume that the same credential will be effective at any future time,
0184 for a couple reasons:

0187 A `[=credential=]` is `<dfn export for="credential">effective</dfn>` for a particular `[=origin=]` if it
0188 is accepted as authentication on that origin. Even if a credential is effective at a particular
0189 point in time, the UA can't assume that the same credential will be effective at any future time,
0190 for a couple reasons:

- 0186 1. A password credential may stop being effective if the account holder changes their password.
- 0187 2. A credential made from a token received over SMS is likely to only be effective for a single
0188 use.

- 0191 1. A password credential may stop being effective if the account holder changes their password.
- 0192 2. A credential made from a token received over SMS is likely to only be effective for a single
0193 use.

0190 Single-use `[=credentials=]` are generated by a `<dfn export>credential source</dfn>`, which could be
0191 a private key, access to a federated account, the ability to receive SMS messages at a particular
0192 phone number, or something else. Credential sources are not exposed to Javascript or explicitly
0193 represented in this specification. To unify the model, we consider a password to be a credential
0194 source on its own, which is simply copied to create password credentials.

0196 Single-use `[=credentials=]` are generated by a `<dfn export>credential source</dfn>`, which could be
0197 a private key, access to a federated account, the ability to receive SMS messages at a particular
0198 phone number, or something else. Credential sources are not exposed to Javascript or explicitly
0199 represented in this specification. To unify the model, we consider a password to be a credential
0200 source on its own, which is simply copied to create password credentials.

0195 Even though the UA can't assume that an effective credential will still be effective if used a

0201 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:

- `<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=]`.
- `<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.
- `<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 unambiguously 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

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:

- `<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=]`.
- `<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.
- `<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 unambiguously 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

0266 " <dfn enum-value for="Credential/[discovery]">remote` </dfn>". The former value means that
0267 all available credential information is stored in the user agent's [=credential store=],
0268 while the latter means that the user agent can discover credentials outside of those
0269 explicitly represented in the [=credential store=] via interaction with some external device
0270 or service.
0271 </dl>
0272
0273 ISSUE: Talk to Tobie/Dominic about the [=interface object=] bits, here and in
0274 [[#algorithm-request]], etc. I'm not sure I've gotten the terminology right. [=interface prototype
0275 object=], maybe?
0276
0277 Some {{Credential}} objects are <dfn for="Credential">origin bound</dfn>: these contain an
0278 internal slot named <dfn for="Credential" attribute>[[origin]]</dfn>, which stores the [=origin=]
0279 for which the {{Credential}} may be [=effective=].
0280
0281 ### `Credential` Internal Methods ### {#credential-internal-methods}
0282
0283 Each [=interface object=] created for interfaces which [=interface/inherit=] from {{Credential}}
0284 defines several internal methods that allow retrieval and storage of {{Credential}} objects:
0285
0286 <section algorithm="collect credentials' internal method">
0287
0288 <dfn for="Credential" method export>[[CollectFromCredentialStore]](origin, options)</dfn> is called
0289 with an [=environment settings object/origin=] and a {{CredentialRequestOptions}}, and returns
0290 a set of {{Credential}} objects from the user
0291 agent's [=credential store=] that match the options provided. If no matching {{Credential}}
0292 objects are available, the returned set will be empty.
0293
0294 <ul class="algorithm">
0295 1. Return an empty set.
0296
0297 </section>
0298
0299 <section algorithm="discover credentials' internal method">
0300
0301 <dfn for="Credential" method export>[[DiscoverFromExternalSource]](origin, options)</dfn> is
0302 called
0303 with an [=environment settings object/origin=] and a {{CredentialRequestOptions}} object.
0304 It returns a {{Credential}} if one can be
0305 returned given the options provided, `null` if no credential is available, or an error if
0306 discovery fails (for example, incorrect options could produce a {{TypeError}}). If this
0307 kind of {{Credential}} is only [=effective=] for a single use or a limited time, this
0308 method is responsible for generating new [=credentials=] using a [=credential source=].
0309
0310 <ul class="algorithm">
0311 1. Return `null`.
0312
0313 </section>
0314
0315 <section algorithm="store a credential' internal method">
0316 <dfn for="Credential" method export>[[Store]](credential)</dfn> is called with a {{Credential}},
0317 and returns once {{Credential}} is persisted to the [=credential store=]:
0318
0319 <ul class="algorithm">
0320 1. Return.

0272 " <dfn enum-value for="Credential/[discovery]">remote` </dfn>". The former value means that
0273 all available credential information is stored in the user agent's [=credential store=],
0274 while the latter means that the user agent can discover credentials outside of those
0275 explicitly represented in the [=credential store=] via interaction with some external device
0276 or service.
0277 </dl>
0278
0279 ISSUE: Talk to Tobie/Dominic about the [=interface object=] bits, here and in
0280 [[#algorithm-request]], etc. I'm not sure I've gotten the terminology right. [=interface prototype
0281 object=], maybe?
0282
0283 Some {{Credential}} objects are <dfn for="Credential">origin bound</dfn>: these contain an
0284 internal slot named <dfn for="Credential" attribute>[[origin]]</dfn>, which stores the [=origin=]
0285 for which the {{Credential}} may be [=effective=].
0286
0287 ### `Credential` Internal Methods ### {#credential-internal-methods}
0288
0289 The {{Credential}} [=interface object=] features several internal methods facilitating
0290 retrieval and storage of {{Credential}} objects, with default "no-op" implementations
0291 as specified in this section, below.
0292
0293 Unless otherwise specified, each [=interface object=] created for interfaces which [=interface/inherit=]
0294 from {{Credential}} MUST provide implementations for at least one of these internal methods,
0295 overriding
0296 {{Credential}}'s default implementations, as appropriate for the [=credential=] type. E.g.,
0297 [[#passwordcredential-interface]] and [[#federatedcredential-interface]].
0298
0299 <h5 id="algorithm-collect-creds" algorithm>[[CollectFromCredentialStore]] internal method</h5>
0300 <dfn for="Credential" method export>[[CollectFromCredentialStore]](origin, options)</dfn> is called
0301 with an [=environment settings object/origin=] and a {{CredentialRequestOptions}}, and returns
0302 a set of {{Credential}} objects from the user
0303 agent's [=credential store=] that match the options provided. If no matching {{Credential}}
0304 objects are available, the returned set will be empty.
0305
0306 {{Credential}}'s default implementation of {{Credential/[[CollectFromCredentialStore]](origin,
0307 options)}}:
0308
0309 <ul class="algorithm">
0310 1. Return an empty set.
0311
0312
0313 <h5 id="algorithm-discover-creds" algorithm>[[DiscoverFromExternalSource]] internal method</h5>
0314 <dfn for="Credential" method export>[[DiscoverFromExternalSource]](origin, options)</dfn> is
0315 called
0316 with an [=environment settings object/origin=] and a {{CredentialRequestOptions}} object.
0317 It returns a {{Credential}} if one can be
0318 returned given the options provided, `null` if no credential is available, or an error if
0319 discovery fails (for example, incorrect options could produce a {{TypeError}}). If this
0320 kind of {{Credential}} is only [=effective=] for a single use or a limited time, this
0321 method is responsible for generating new [=credentials=] using a [=credential source=].
0322
0323 {{Credential}}'s default implementation of {{Credential/[[DiscoverFromExternalSource]](origin,
0324 options)}}:
0325
0326 <ul class="algorithm">
0327 1. Return `null`.
0328
0329
0330 <h5 id="algorithm-store-cred" algorithm>[[Store]] internal method</h5>
0331 <dfn for="Credential" method export>[[Store]](credential)</dfn> is called with a {{Credential}},
0332 and returns once {{Credential}} is persisted to the [=credential store=].
0333
0334 {{Credential}}'s default implementation of {{Credential/[[Store]](credential)}}:
0335
0336 <ul class="algorithm">
0337 1. Return.

```

0318 </ul>
0319 </section>
0320
0321 <section algorithm="create a credential' internal method">
0322 <dfn for="Credential" method export>[[Create]](options)</dfn> is called with a
0323 {{CredentialCreationOptions}}, and returns either a {{Credential}}, if one can be created,
0324 null if no credential was created, or an error if creation fails due to exceptional situations
0325 (for example, incorrect options could produce a {{TypeError}}):

```

```

0326 <ul class="algorithm">
0327 1. Return `null`.
0328 </ul>
0329 </section>
0330
0331 Unless otherwise specified, the [=interface objects=] for [=interfaces=] which [=interface/inherit=] from
0332 {{Credential}} will alias {{Credential}}'s implementation of these three internal methods. Since that
0333 implementation isn't particularly useful, derived interfaces MUST override one or the other.
0334

```

```

0335 ### `CredentialUserData` Mixin ### {#credentialuserdata-mixin}
0336
0337 Some {{Credential}} objects contain data which aims to give users a human-readable disambiguation
0338 mechanism in the [=credential chooser=] by providing a friendly name and icon:
0339
0340 <pre class="idl">
0341 [NoInterfaceObject, SecureContext]
0342 interface CredentialUserData {
0343   readonly attribute USVString name;
0344   readonly attribute USVString iconURL;
0345 };
0346 </pre>
0347 <dl dfn-for="CredentialUserData">
0348 : <dfn attribute>name</dfn>
0349 :: A name associated with the credential, intended as a human-understandable public name for
0350 display in a [=credential chooser=].
0351 : <dfn attribute>iconURL</dfn>
0352 :: A URL pointing to an image for the credential, intended for display in a
0353 [=credential chooser=]. This URL MUST be an <a><i lang="la">a priori</i> authenticated
0354 URL</a>.
0355 </dl>
0356
0357 ## `navigator.credentials` ## {#framework-credential-management}
0358
0359 Developers retrieve {{Credential}}s and interact with the user agent's [=credential store=] via
0360 methods exposed on the
0361 {{CredentialsContainer}} interface, which hangs off the {{Navigator}} object as
0362 <a attribute for="Navigator" It="credentials"> navigator.credentials </a>.
0363

```

```

0335 </ul>
0336
0337 <h5 id="algorithm-create-cred" algorithm> [[Create]] internal method</h5>
0338 <dfn for="Credential" method export>[[Create]](origin, options)</dfn> is called
0339 [=in parallel=] with an [=environment settings object/origin=] and a {{CredentialCreationOptions}},
0340 and either:
0341
0342 * creates a {{Credential}}, or,
0343 * does not create a credential and returns `null`, or,
0344 * returns an error if creation fails due to exceptional situations
0345 (for example, incorrect options could produce a {{TypeError}}).
0346
0347 When creating a {{Credential}}, it will either:
0348
0349 * directly return a {{Credential}} (in which case it accomplished everything while
0350 running [=in parallel=]), or,
0351 * return a [=constructCredential=] algorithm which yields a {{Credential}} when
0352 subsequently invoked from a [=task=].
0353
0354 {{Credential}}'s default implementation of {{Credential/[[Create]](origin, options)}}:
0355
0356 <ul class="algorithm">
0357 1. Return `null`.
0358 </ul>
0359
0360 <h6 id="algorithm-construct-cred">constructCredential Algorithm</h6>
0361
0362 A <dfn>constructCredential</dfn> algorithm is a [=queue=] of steps
0363 to be invoked by a [=task=] from the [=DOM manipulation task source=].
0364 [=constructCredential=]'s purpose is to construct an [=interface object=] derived
0365 from {{Credential}}, given a [=current settings object=]'s
0366 [=environment settings object/global object=].
0367
0368 Note: <code>[=constructCredential=]</code>'s algorithm steps are defined on a
0369 per-[=credential=] type basis by those [=credential=] type specifications needing to
0370 employ it. E.g., [[WEBAUTHN]].
0371
0372 ### `CredentialUserData` Mixin ### {#credentialuserdata-mixin}
0373
0374 Some {{Credential}} objects contain data which aims to give users a human-readable disambiguation
0375 mechanism in the [=credential chooser=] by providing a friendly name and icon:
0376
0377 <pre class="idl">
0378 [NoInterfaceObject, SecureContext]
0379 interface CredentialUserData {
0380   readonly attribute USVString name;
0381   readonly attribute USVString iconURL;
0382 };
0383 </pre>
0384 <dl dfn-for="CredentialUserData">
0385 : <dfn attribute>name</dfn>
0386 :: A name associated with the credential, intended as a human-understandable public name for
0387 display in a [=credential chooser=].
0388 : <dfn attribute>iconURL</dfn>
0389 :: A URL pointing to an image for the credential, intended for display in a
0390 [=credential chooser=]. This URL MUST be an <a><i lang="la">a priori</i> authenticated
0391 URL</a>.
0392 </dl>
0393
0394 ## `navigator.credentials` ## {#framework-credential-management}
0395
0396 Developers retrieve {{Credential}}s and interact with the user agent's [=credential store=] via
0397 methods exposed on the
0398 {{CredentialsContainer}} interface, which hangs off the {{Navigator}} object as
0399 <a attribute for="Navigator" It="credentials"> navigator.credentials </a>.
0400
0401

```

```

0364 <pre class="idl">
0365   partial interface Navigator {
0366     [SecureContext, SameObject] readonly attribute CredentialsContainer credentials;
0366   };
0366 </pre>
0370
0371 The <dfn for="Navigator" attribute>\`credentials`</dfn> attribute MUST return the
0372 {{CredentialsContainer}} associated with the [=context object=].
0373
0374 Note: As discussed in [[#insecure-sites]], the credential management API is exposed only in
0375 [=Secure Contexts=].
0376
0377 <pre class="idl">
0378   [Exposed=Window, SecureContext]
0379   interface CredentialsContainer {
0380     Promise<Credential?> get(optional CredentialRequestOptions options);
0381     Promise<Credential> store(Credential credential);
0382     Promise<Credential?> create(optional CredentialCreationOptions options);
0383     Promise<void> preventSilentAccess();
0384   };
0385
0386   dictionary CredentialData {
0387     required USVString id;
0388   };
0389 </pre>
0390
0391 <dl dfn-for="CredentialsContainer">
0392   : <dfn method>get(options)</dfn>
0393   :: When {{CredentialsContainer/get()}} is called, the user agent MUST return the result of
0394     executing <a abstract-op>Request a `Credential`</a> on
0395     {{CredentialsContainer/get(options)}}.
0396
0397     <pre class="argumentdef" for="CredentialsContainer/get(options)">
0398       options: The set of properties governing the scope of the request.
0399     </pre>
0400
0401   : <dfn method It="store(credential)|store()">store(credential)</dfn>
0402   :: When {{CredentialsContainer/store()}} is called, the user agent MUST return the result of
0403     executing <a abstract-op>Store a `Credential`</a> on
0404     {{CredentialsContainer/store(credential)/credential}}.
0405
0406     <pre class="argumentdef" for="CredentialsContainer/store(credential)">
0407       credential: The credential to be stored.
0408     </pre>
0409
0410   : <dfn method It="create(options)|create()">create(options)</dfn>
0411   :: When {{CredentialsContainer/create()}} is called, the user agent MUST return the result of
0412     executing <a abstract-op>Create a `Credential`</a> on
0413     {{CredentialsContainer/create(options)/options}}.
0414
0415     <pre class="argumentdef" for="CredentialsContainer/create(options)">
0416       options: The options used to create a `Credential`.
0417     </pre>
0418
0419   : <dfn method>preventSilentAccess()</dfn>
0420   :: When {{CredentialsContainer/preventSilentAccess()}} is called, the user agent MUST return
0421     the result of executing <a abstract-op>Prevent Silent Access</a> on the <a>current settings
0422     object</a>.
0423
0424     Note: The intent here is a signal from the origin that the user has signed out. That
0425     is, after a click on a "Sign out" button, the site updates the user's session info, and
0426     calls `navigator.credentials.preventSilentAccess()`. This sets the <a>prevent silent
0427     access` flag</a>, meaning that credentials will not be automatically handed back to the
0428     page next time the user visits.
0429
0430     Note: This function was previously called `requireUserMediation()` which should be considered
0431     deprecated.
0432 </dl>

```

```

0402 <pre class="idl">
0403   partial interface Navigator {
0404     [SecureContext, SameObject] readonly attribute CredentialsContainer credentials;
0405   };
0406 </pre>
0407
0408 The <dfn for="Navigator" attribute>\`credentials`</dfn> attribute MUST return the
0409 {{CredentialsContainer}} associated with the [=context object=].
0410
0411 Note: As discussed in [[#insecure-sites]], the credential management API is exposed only in
0412 [=Secure Contexts=].
0413
0414 <pre class="idl">
0415   [Exposed=Window, SecureContext]
0416   interface CredentialsContainer {
0417     Promise<Credential?> get(optional CredentialRequestOptions options);
0418     Promise<Credential> store(Credential credential);
0419     Promise<Credential?> create(optional CredentialCreationOptions options);
0420     Promise<void> preventSilentAccess();
0421   };
0422
0423   dictionary CredentialData {
0424     required USVString id;
0425   };
0426 </pre>
0427
0428 <dl dfn-for="CredentialsContainer">
0429   : <dfn method>get(options)</dfn>
0430   :: When {{CredentialsContainer/get()}} is called, the user agent MUST return the result of
0431     executing <a abstract-op>Request a `Credential`</a> on
0432     {{CredentialsContainer/get(options)}}.
0433
0434     <pre class="argumentdef" for="CredentialsContainer/get(options)">
0435       options: The set of properties governing the scope of the request.
0436     </pre>
0437
0438   : <dfn method It="store(credential)|store()">store(credential)</dfn>
0439   :: When {{CredentialsContainer/store()}} is called, the user agent MUST return the result of
0440     executing <a abstract-op>Store a `Credential`</a> on
0441     {{CredentialsContainer/store(credential)/credential}}.
0442
0443     <pre class="argumentdef" for="CredentialsContainer/store(credential)">
0444       credential: The credential to be stored.
0445     </pre>
0446
0447   : <dfn method It="create(options)|create()">create(options)</dfn>
0448   :: When {{CredentialsContainer/create()}} is called, the user agent MUST return the result of
0449     executing <a abstract-op>Create a `Credential`</a> on
0450     {{CredentialsContainer/create(options)/options}}.
0451
0452     <pre class="argumentdef" for="CredentialsContainer/create(options)">
0453       options: The options used to create a `Credential`.
0454     </pre>
0455
0456   : <dfn method>preventSilentAccess()</dfn>
0457   :: When {{CredentialsContainer/preventSilentAccess()}} is called, the user agent MUST return
0458     the result of executing <a abstract-op>Prevent Silent Access</a> on the <a>current settings
0459     object</a>.
0460
0461     Note: The intent here is a signal from the origin that the user has signed out. That
0462     is, after a click on a "Sign out" button, the site updates the user's session info, and
0463     calls `navigator.credentials.preventSilentAccess()`. This sets the <a>prevent silent
0464     access` flag</a>, meaning that credentials will not be automatically handed back to the
0465     page next time the user visits.
0466
0467     Note: This function was previously called `requireUserMediation()` which should be considered
0468     deprecated.
0469 </dl>

```

```

0433 <div algorithm="create credentialscontainer">
0434   When a {{Navigator}} object (InavigatorI) is created, the user agent MUST create a
0435   new {{CredentialsContainer}} object, using InavigatorI's [=relevant Realm=], and associate it
0436   with InavigatorI.
0437 </div>
0438
0439 ### The `CredentialRequestOptions` Dictionary ### {#credentialrequestoptions-dictionary}
0440
0441 In order to retrieve a {{Credential}} via {{CredentialsContainer/get()}}, the caller specifies a
0442 few parameters in a {{CredentialRequestOptions}} object.
0443
0444 Note: The {{CredentialRequestOptions}} dictionary is an extension point. If and when new types of
0445 credentials are introduced that require options, their dictionary types will be added to the
0446 dictionary so they can be passed into the request. See [[#implementation-extension]].
0447
0448 <pre class="idl">
0449   dictionary CredentialRequestOptions {
0450     CredentialMediationRequirement mediation = "optional";
0451   };
0452 </pre>
0453 <dl dfn-for="CredentialRequestOptions" dfn-type="dict-member">
0454   : <dfn>mediation</dfn>
0455   :: This property specifies the mediation requirements for a given credential request. The
0456     meaning of each enum value is described below in {{CredentialMediationRequirement}}.
0457     Processing details are defined in [[#algorithm-request]].
0458 </dl>
0459
0460 <div class="note">
0461   Earlier versions of this specification defined a boolean `unmediated` member. Setting that
0462   to `true` had the effect of setting {{CredentialRequestOptions/mediation}} to
0463   "{{CredentialMediationRequirement/silent}}", and setting it to `false` had the effect of
0464   setting {{CredentialRequestOptions/mediation}} to "{{CredentialMediationRequirement/optional}}".
0465
0466   `unmediated` should be considered deprecated; new code should instead rely on
0467   {{CredentialRequestOptions/mediation}}.
0468 </div>
0469
0470 <div algorithm="relevant credential interfaces">
0471   The <dfn for="CredentialRequestOptions">relevant credential interface objects</dfn> for a given
0472   {{CredentialRequestOptions}} (loptionsI) is a set of [=interface objects=], collected as follows:
0473
0474   <ol>
0475     1. Let lsettingsI be the <a>current settings object</a>
0476
0477     2. Let linterface objectsI be the set of [=interface objects=] on lsettingsI'
0478       [=environment settings object/global object=].
0479
0480     3. Let lrelevant interface objectsI be an empty set.
0481
0482     4. For each lobjectI in linterface objectsI:
0483
0484       1. If lobjectI's [=inherited interfaces=] do not <a for="set">contain</a> {{Credential}},
0485         <a for="iteration">continue</a>.
0486
0487       2. Let lkeyI be lobjectI's {{Credential/[[type]]}} slot's value.
0488
0489       3. If loptionsI[lkeyI] <a for="map">exists</a>, <a for="set">append</a> lobjectI to
0490         lrelevant interface objectsI.
0491   </ol>
0492
0493   ISSUE: jyasskin@ suggests replacing the iteration through the interface objects with a registry.
0494   I'm not sure which is less clear, honestly. I'll leave it like this for the moment, and we can
0495   argue about whether this is too much of a `COMEFROM` interface.
0496 </div>
0497
0498 <div algorithm="can collect locally">
0499   A given {{CredentialRequestOptions}} loptionsI is
0500   <dfn for="CredentialRequestOptions">matchable <i lang="la">a priori</i></dfn> if the following
0501

```

```

0471 <div algorithm="create credentialscontainer">
0472   When a {{Navigator}} object (InavigatorI) is created, the user agent MUST create a
0473   new {{CredentialsContainer}} object, using InavigatorI's [=relevant Realm=], and associate it
0474   with InavigatorI.
0475 </div>
0476
0477 ### The `CredentialRequestOptions` Dictionary ### {#credentialrequestoptions-dictionary}
0478
0479 In order to retrieve a {{Credential}} via {{CredentialsContainer/get()}}, the caller specifies a
0480 few parameters in a {{CredentialRequestOptions}} object.
0481
0482 Note: The {{CredentialRequestOptions}} dictionary is an extension point. If and when new types of
0483 credentials are introduced that require options, their dictionary types will be added to the
0484 dictionary so they can be passed into the request. See [[#implementation-extension]].
0485
0486 <pre class="idl">
0487   dictionary CredentialRequestOptions {
0488     CredentialMediationRequirement mediation = "optional";
0489   };
0490 </pre>
0491 <dl dfn-for="CredentialRequestOptions" dfn-type="dict-member">
0492   : <dfn>mediation</dfn>
0493   :: This property specifies the mediation requirements for a given credential request. The
0494     meaning of each enum value is described below in {{CredentialMediationRequirement}}.
0495     Processing details are defined in [[#algorithm-request]].
0496 </dl>
0497
0498 <div class="note">
0499   Earlier versions of this specification defined a boolean `unmediated` member. Setting that
0500   to `true` had the effect of setting {{CredentialRequestOptions/mediation}} to
0501   "{{CredentialMediationRequirement/silent}}", and setting it to `false` had the effect of
0502   setting {{CredentialRequestOptions/mediation}} to "{{CredentialMediationRequirement/optional}}".
0503
0504   `unmediated` should be considered deprecated; new code should instead rely on
0505   {{CredentialRequestOptions/mediation}}.
0506 </div>
0507
0508 <div algorithm="relevant credential interfaces">
0509   The <dfn for="CredentialRequestOptions">relevant credential interface objects</dfn> for a given
0510   {{CredentialRequestOptions}} (loptionsI) is a set of [=interface objects=], collected as follows:
0511
0512   <ol>
0513     1. Let lsettingsI be the <a>current settings object</a>
0514
0515     2. Let linterface objectsI be the set of [=interface objects=] on lsettingsI'
0516       [=environment settings object/global object=].
0517
0518     3. Let lrelevant interface objectsI be an empty set.
0519
0520     4. For each lobjectI in linterface objectsI:
0521
0522       1. If lobjectI's [=inherited interfaces=] do not <a for="set">contain</a> {{Credential}},
0523         <a for="iteration">continue</a>.
0524
0525       2. Let lkeyI be lobjectI's {{Credential/[[type]]}} slot's value.
0526
0527       3. If loptionsI[lkeyI] <a for="map">exists</a>, <a for="set">append</a> lobjectI to
0528         lrelevant interface objectsI.
0529   </ol>
0530
0531   ISSUE: jyasskin@ suggests replacing the iteration through the interface objects with a registry.
0532   I'm not sure which is less clear, honestly. I'll leave it like this for the moment, and we can
0533   argue about whether this is too much of a `COMEFROM` interface.
0534 </div>
0535
0536 <div algorithm="can collect locally">
0537   A given {{CredentialRequestOptions}} loptionsI is
0538   <dfn for="CredentialRequestOptions">matchable <i lang="la">a priori</i></dfn> if the following
0539

```

```
0502 steps return `true`:  
0503  
0504 <ol>  
0505 1. For each linterfacel in loptions! [=relevant credential interface objects=]:  
0506  
0507 1. If linterfacel's {{Credential/[discovery]}} slot's value is not  
0508 ">{{Credential/[discovery]/credential store}}", return `false`.  
0509  
0510 2. Return `true`.  
0511 </ol>  
0512  
0513 Note: When executing {{get(options)}}, we only return credentials without [=user mediation=] if  
0514 the provided {{CredentialRequestOptions}} is <a>matchable <i lang="la">a priori</i></a>. If any  
0515 credential types are requested that could require discovery from some external service (OAuth  
0516 tokens, security key authenticators, etc.), then [=user mediation=] will be required in order  
0517 to guide the discovery process (by choosing a federated identity provider, BTLE device, etc).  
0518 </div>  
0519  
0520 ### Mediation Requirements ### {#mediation-requirements}  
0521  
0522 When making a request via {{get(options)}}, developers can set a case-by-case requirement for  
0523 [=user mediation=] by choosing the appropriate {{CredentialMediationRequirement}} enum value.  
0524  
0525 Note: The [#user-mediation] section gives more detail on the concept in general, and its  
0526 implications on how the user agent deals with individual requests for a given origin).  
0527  
0528 <pre class="idl">  
0529 enum CredentialMediationRequirement {  
0530 "silent",  
0531 "optional",  
0532 "required"  
0533 };  
0534 </pre>  
0535 <dl dfn-for="CredentialMediationRequirement" dfn-type="enum-value">  
0536 : <dfn>silent</dfn>  
0537 :: User mediation is suppressed for the given operation. If the operation can be performed  
0538 without user involvement, wonderful. If user involvement is necessary, then the operation  
0539 will return `null` rather than involving the user.  
0540  
0541 Note: The intended usage is to support ["Keep me signed-into this site"  
0542 scenarios](#example-mediation-silent), where a developer may wish to silently obtain  
0543 credentials if a user should be automatically signed in, but to delay bothering the user  
0544 with a sign-in prompt until they actively choose to sign-in.  
0545  
0546 : <dfn>optional</dfn>  
0547 :: If credentials can be handed over for a given operation without user mediation, they will  
0548 be. If [=requires user mediation/user mediation is required=], then the user agent will  
0549 involve the user in the decision.  
0550  
0551 Note: This is the default behavior for {{get()}}, and is intended to serve a case where a  
0552 developer has reasonable confidence that a user expects to start a sign-in operation. If  
0553 a user has just clicked "sign-in" for example, then they won't be surprised or confused to  
0554 see a [=credential chooser=] if necessary.  
0555  
0556 : <dfn>required</dfn>  
0557 :: The user agent will not hand over credentials without [=user mediation=], even if the  
0558 [=prevent silent access flag=] is unset for an origin.  
0559  
0560 Note: This requirement is intended to support [reauthentication](#example-mediation-require)  
0561 or [user-switching](#example-mediation-switch) scenarios. Further, the requirement is tied  
0562 to a specific operation, and does not affect the [=prevent silent access flag=] for the  
0563 origin. To set that flag, developers should call {{preventSilentAccess()}}.  
0564 </dl>  
0565  
0566 ##### Examples ##### {#mediation-examples}  
0567  
0568 <div class="example" id="example-mediation-silent">  
0569 MegaCorp, Inc. wishes to seamlessly sign in users when possible. They can do so by calling  
0570 {{get()}} for all non-signed in users at some convinient point while a landing page is loading,
```

```
0540 steps return `true`:  
0541  
0542 <ol>  
0543 1. For each linterfacel in loptions! [=relevant credential interface objects=]:  
0544  
0545 1. If linterfacel's {{Credential/[discovery]}} slot's value is not  
0546 ">{{Credential/[discovery]/credential store}}", return `false`.  
0547  
0548 2. Return `true`.  
0549 </ol>  
0550  
0551 Note: When executing {{get(options)}}, we only return credentials without [=user mediation=] if  
0552 the provided {{CredentialRequestOptions}} is <a>matchable <i lang="la">a priori</i></a>. If any  
0553 credential types are requested that could require discovery from some external service (OAuth  
0554 tokens, security key authenticators, etc.), then [=user mediation=] will be required in order  
0555 to guide the discovery process (by choosing a federated identity provider, BTLE device, etc).  
0556 </div>  
0557  
0558 ### Mediation Requirements ### {#mediation-requirements}  
0559  
0560 When making a request via {{get(options)}}, developers can set a case-by-case requirement for  
0561 [=user mediation=] by choosing the appropriate {{CredentialMediationRequirement}} enum value.  
0562  
0563 Note: The [#user-mediation] section gives more detail on the concept in general, and its  
0564 implications on how the user agent deals with individual requests for a given origin).  
0565  
0566 <pre class="idl">  
0567 enum CredentialMediationRequirement {  
0568 "silent",  
0569 "optional",  
0570 "required"  
0571 };  
0572 </pre>  
0573 <dl dfn-for="CredentialMediationRequirement" dfn-type="enum-value">  
0574 : <dfn>silent</dfn>  
0575 :: User mediation is suppressed for the given operation. If the operation can be performed  
0576 without user involvement, wonderful. If user involvement is necessary, then the operation  
0577 will return `null` rather than involving the user.  
0578  
0579 Note: The intended usage is to support ["Keep me signed-into this site"  
0580 scenarios](#example-mediation-silent), where a developer may wish to silently obtain  
0581 credentials if a user should be automatically signed in, but to delay bothering the user  
0582 with a sign-in prompt until they actively choose to sign-in.  
0583  
0584 : <dfn>optional</dfn>  
0585 :: If credentials can be handed over for a given operation without user mediation, they will  
0586 be. If [=requires user mediation/user mediation is required=], then the user agent will  
0587 involve the user in the decision.  
0588  
0589 Note: This is the default behavior for {{get()}}, and is intended to serve a case where a  
0590 developer has reasonable confidence that a user expects to start a sign-in operation. If  
0591 a user has just clicked "sign-in" for example, then they won't be surprised or confused to  
0592 see a [=credential chooser=] if necessary.  
0593  
0594 : <dfn>required</dfn>  
0595 :: The user agent will not hand over credentials without [=user mediation=], even if the  
0596 [=prevent silent access flag=] is unset for an origin.  
0597  
0598 Note: This requirement is intended to support [reauthentication](#example-mediation-require)  
0599 or [user-switching](#example-mediation-switch) scenarios. Further, the requirement is tied  
0600 to a specific operation, and does not affect the [=prevent silent access flag=] for the  
0601 origin. To set that flag, developers should call {{preventSilentAccess()}}.  
0602 </dl>  
0603  
0604 ##### Examples ##### {#mediation-examples}  
0605  
0606 <div class="example" id="example-mediation-silent">  
0607 MegaCorp, Inc. wishes to seamlessly sign in users when possible. They can do so by calling  
0608 {{get()}} for all non-signed in users at some convinient point while a landing page is loading,
```

```

0571 passing in a {{CredentialRequestOptions/mediation}} member set to
0572 "{{CredentialMediationRequirement/silent}}". This ensures that users who have opted-into
0573 dropping the requirements for user mediation (as described in [[#user-mediation-requirement]])
0574 are signed in, and users who haven't opted-into such behavior won't be bothered with a confusing
0575 [=credential chooser=] popping up without context:
0576
0577 <pre>
0578 window.addEventListener('load', => {
0579   var c = await navigator.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer"
0580   method lt="get()">get</a>({
0581     <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a
0582     for="CredentialMediationRequirement" enum-value>silent</a>'
0583   });
0584   if (c) {
0585     // Hooray! Let's sign the user in using these credentials!
0586   };
0587 </pre>
0588 </div>
0589
0590 <div class="example" id="example-mediation-optional">
0591   When a user clicks "Sign In", MegaCorp, Inc. wishes to give them the smoothest possible
0592   experience. If they have [[#user-mediation-requirementlopted-into]] signing in without
0593   [=user mediation=], and the user agent can unambiguously choose a credential, great! If
0594   not, a [=credential chooser=] will be presented.
0595
0596   <pre>
0597   document.querySelector('#sign-in').addEventListener('click', e => {
0598     var c = await navigator.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer"
0599     method lt="get()">get</a>({
0600     <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a
0601     for="CredentialMediationRequirement" enum-value>optional</a>'
0602   });
0603   if (c) {
0604     // Hooray! Let's sign the user in using these credentials!
0605   };
0606 </pre>
0607
0608   Note: MegaCorp, Inc. could also have left off the {{CredentialRequestOptions/mediation}}
0609   member entirely, as "{{CredentialMediationRequirement/optional}}" is its default.
0610 </div>
0611
0612 <div class="example" id="example-mediation-require">
0613   MegaCorp, Inc. wishes to protect a sensitive operation by requiring a user to reauthenticate
0614   before taking action. Even if a user has [[#user-mediation-requirementlopted-into]] signing in
0615   without [=user mediation=], MegaCorp, Inc. can ensure that the user agent requires mediation
0616   by calling {{get()}} with a {{CredentialRequestOptions/mediation}} member set to
0617   "{{CredentialMediationRequirement/required}}":
0618
0619   Note: Depending on the security model of the browser or the credential type, this may require
0620   the user to authenticate themselves in some way, perhaps by entering a master password, scanning
0621   a fingerprint, etc. before a credential is handed to the website.
0622
0623   <pre>
0624   document.querySelector('#important-form').addEventListener('submit', e => {
0625     var c = await navigator.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer"
0626     method lt="get()">get</a>({
0627     <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a
0628     for="CredentialMediationRequirement" enum-value>required</a>'
0629   });
0630   if (c) {
0631     // Verify that lcl enables access, and cancel the submission
0632     // if it doesn't.
0633     } else {
0634       e.preventDefault();

```

```

0609 passing in a {{CredentialRequestOptions/mediation}} member set to
0610 "{{CredentialMediationRequirement/silent}}". This ensures that users who have opted-into
0611 dropping the requirements for user mediation (as described in [[#user-mediation-requirement]])
0612 are signed in, and users who haven't opted-into such behavior won't be bothered with a confusing
0613 [=credential chooser=] popping up without context:
0614
0615 <pre>
0616 window.addEventListener('load', => {
0617   var c = await navigator.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer"
0618   method lt="get()">get</a>({
0619     <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a
0620     for="CredentialMediationRequirement" enum-value>silent</a>'
0621   });
0622   if (c) {
0623     // Hooray! Let's sign the user in using these credentials!
0624   };
0625 </pre>
0626 </div>
0627
0628 <div class="example" id="example-mediation-optional">
0629   When a user clicks "Sign In", MegaCorp, Inc. wishes to give them the smoothest possible
0630   experience. If they have [[#user-mediation-requirementlopted-into]] signing in without
0631   [=user mediation=], and the user agent can unambiguously choose a credential, great! If
0632   not, a [=credential chooser=] will be presented.
0633
0634   <pre>
0635   document.querySelector('#sign-in').addEventListener('click', e => {
0636     var c = await navigator.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer"
0637     method lt="get()">get</a>({
0638     <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a
0639     for="CredentialMediationRequirement" enum-value>optional</a>'
0640   });
0641   if (c) {
0642     // Hooray! Let's sign the user in using these credentials!
0643   };
0644 </pre>
0645
0646   Note: MegaCorp, Inc. could also have left off the {{CredentialRequestOptions/mediation}}
0647   member entirely, as "{{CredentialMediationRequirement/optional}}" is its default.
0648 </div>
0649
0650 <div class="example" id="example-mediation-require">
0651   MegaCorp, Inc. wishes to protect a sensitive operation by requiring a user to reauthenticate
0652   before taking action. Even if a user has [[#user-mediation-requirementlopted-into]] signing in
0653   without [=user mediation=], MegaCorp, Inc. can ensure that the user agent requires mediation
0654   by calling {{get()}} with a {{CredentialRequestOptions/mediation}} member set to
0655   "{{CredentialMediationRequirement/required}}":
0656
0657   Note: Depending on the security model of the browser or the credential type, this may require
0658   the user to authenticate themselves in some way, perhaps by entering a master password, scanning
0659   a fingerprint, etc. before a credential is handed to the website.
0660
0661   <pre>
0662   document.querySelector('#important-form').addEventListener('submit', e => {
0663     var c = await navigator.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer"
0664     method lt="get()">get</a>({
0665     <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a
0666     for="CredentialMediationRequirement" enum-value>required</a>'
0667   });
0668   if (c) {
0669     // Verify that lcl enables access, and cancel the submission
0670     // if it doesn't.
0671     } else {
0672       e.preventDefault();

```

```
0634     });
0635   });
0636 </pre>
0637 </div>
0638
0639 <div class="example" id="example-mediation-switch">
0640   MegaCorp, Inc. wishes to support signing into multiple user accounts at once. In order to ensure
0641   that the user gets a chance to select a different credential, MegaCorp, Inc. can call {{get()}}
0642   with a {{CredentialRequestOptions/mediation}} member set to
0643   "{{CredentialMediationRequirement/required}}" in order to ensure that that credentials aren't
0644   returned automatically in response to clicking on an "Add account" button:
0645
0646   <pre>
0647     document.querySelector('#switch-button').addEventListener('click', e =>{
0648       var c = await navigator.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer"
0649     method lt="get()">get</a>({
0650       ****
0651       <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a
0652     for="CredentialMediationRequirement" enum-value>required</a>'
0653     });
0654     if (c) {
0655       // Sign the user in using lcl.
0656     }
0657   </pre>
0658 </div>
0659 ## The `CredentialCreationOptions` Dictionary ## {#credentialcreationoptions-dictionary}
0660
0661 In order to create a {{Credential}} via {{CredentialsContainer/create()}}, the caller specifies a
0662 few parameters in a {{CredentialCreationOptions}} object.
0663
0664 Note: The {{CredentialCreationOptions}} dictionary is an extension point. If and when new types of
0665 credentials are introduced, they will add to the dictionary so they can be passed into the
0666 creation method. See [[#implementation-extension]], and the extensions introduced in this document:
0667 [[#passwordcredential-interface]] and [[#federatedcredential-interface]].
0668
0669 <pre class="idl">
0670   dictionary CredentialCreationOptions {
0671   };
0672 </pre>
0673
0674 ## Algorithms ## {#algorithms}
0675
0676 <h4 id="algorithm-request" algorithm>Request a `Credential` </h4>
0677
0678 The <dfn abstract-op>Request a `Credential` </dfn> algorithm accepts a {{CredentialRequestOptions}}
0679 (loptions), and returns a {{Promise}} that resolves with a {{Credential}} if one can be
0680 unambiguously obtained, or with `null` if not.
0681
0682 <ol class="algorithm">
0683   1. Let lsettings be the <a>current settings object</a>
0684
0685   2. Assert: lsettings is a [=secure context=].
0686
0687   3. Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
0688   are true:
0689     1. lsettings does not have a [=environment settings object/responsible document=].
0690     2. lsettings' [=environment settings object/responsible document=] is not the
0691     [=active document=] in a [=top-level browsing context=].
0692
0693   4. Let lpl be [=a new promise=].
0694
0695   5. Let lorigin be the [=current settings object=]'s [=environment settings object/origin=].
0696
0697   6. Run the following steps [=in parallel=]:
```

```
0672     });
0673   });
0674 </pre>
0675 </div>
0676
0677 <div class="example" id="example-mediation-switch">
0678   MegaCorp, Inc. wishes to support signing into multiple user accounts at once. In order to ensure
0679   that the user gets a chance to select a different credential, MegaCorp, Inc. can call {{get()}}
0680   with a {{CredentialRequestOptions/mediation}} member set to
0681   "{{CredentialMediationRequirement/required}}" in order to ensure that that credentials aren't
0682   returned automatically in response to clicking on an "Add account" button:
0683
0684   <pre>
0685     document.querySelector('#switch-button').addEventListener('click', e =>{
0686       var c = await navigator.<a for="Navigator" attribute>credentials</a>.<a for="CredentialsContainer"
0687     method lt="get()">get</a>({
0688       ****
0689       <a for="CredentialRequestOptions" dict-member>mediation</a>: '<a
0690     for="CredentialMediationRequirement" enum-value>required</a>'
0691     });
0692     if (c) {
0693       // Sign the user in using lcl.
0694     }
0695   </pre>
0696 </div>
0697 ## The `CredentialCreationOptions` Dictionary ## {#credentialcreationoptions-dictionary}
0698
0699 In order to create a {{Credential}} via {{CredentialsContainer/create()}}, the caller specifies a
0700 few parameters in a {{CredentialCreationOptions}} object.
0701
0702 Note: The {{CredentialCreationOptions}} dictionary is an extension point. If and when new types of
0703 credentials are introduced, they will add to the dictionary so they can be passed into the
0704 creation method. See [[#implementation-extension]], and the extensions introduced in this document:
0705 [[#passwordcredential-interface]] and [[#federatedcredential-interface]].
0706
0707 <pre class="idl">
0708   dictionary CredentialCreationOptions {
0709   };
0710 </pre>
0711
0712 ## Algorithms ## {#algorithms}
0713
0714 <h4 id="algorithm-request" algorithm>Request a `Credential` </h4>
0715
0716 The <dfn abstract-op>Request a `Credential` </dfn> algorithm accepts a {{CredentialRequestOptions}}
0717 (loptions), and returns a {{Promise}} that resolves with a {{Credential}} if one can be
0718 unambiguously obtained, or with `null` if not.
0719
0720 <ol class="algorithm">
0721   1. Let lsettings be the <a>current settings object</a>
0722
0723   2. Assert: lsettings is a [=secure context=].
0724
0725   3. Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
0726   are true:
0727     1. lsettings does not have a [=environment settings object/responsible document=].
0728     2. lsettings' [=environment settings object/responsible document=] is not the
0729     [=active document=] in a [=top-level browsing context=].
0730
0731   4. Let lpl be [=a new promise=].
0732
0733   5. Let lorigin be the [=current settings object=]'s [=environment settings object/origin=].
0734
0735   6. Run the following steps [=in parallel=]:
```

```
0701 1. Let IcredentialsI be the result of <a abstract-op It="collect local">collecting
0702   `Credential`'s from the credential store</a>, given IoriginI and IoptionsI.
0703
0704 2. If IcredentialsI is an [=exception=], [=reject=] Ipl with IcredentialsI.
0705
0706 3. If all of the following statements are true, resolve Ipl with IcredentialsI[0] and
0707   skip the remaining steps:
0708
0709   1. IcredentialsI' [=set/size=] is 1
0710
0711   2. IoriginI does not [=origin/requires user mediation]require user mediation=]
0712
0713   3. IoptionsI is <a>matchable <i lang="la">a priori</i></a>.
0714
0715   4. IoptionsI.{{CredentialRequestOptions/mediation}} is not
0716     "{{CredentialMediationRequirement/required}}".
0717
0718   ISSUE: This might be the wrong model. It would be nice to support a site that wished
0719   to accept either username/passwords or webauthn-style credentials without forcing
0720   a chooser for those users who use the former, and who wish to remain signed in.
0721
0722 4. If IoptionsI' {{CredentialRequestOptions/mediation}} is
0723   "{{CredentialMediationRequirement/silent}}", [=resolve=] Ipl with `null`, and skip
0724   the remaining steps.
0725
0726 5. Let IchoiceI be the result of <a abstract-op It="ask to choose">asking the user to
0727   choose a `Credential` </a>, given IoptionsI and IcredentialsI.
0728
0729 6. If IchoiceI is `null` or a {{Credential}}, [=resolve=] Ipl with IchoiceI and skip the
0730   remaining steps.
0731
0732 7. Assert: IchoiceI is an [=interface object=].
0733
0734 8. Let IresultI be the result of executing IchoiceI's
0735   {{{DiscoverFromExternalSource}}}(origin, options)}, given IoriginI and IoptionsI.
0736
0737 9. If IresultI is a {{Credential}} or `null`, resolve Ipl with IresultI.
0738
0739   Otherwise, [=reject=] Ipl with IresultI.
0740
0741 7. Return Ipl.
0742 </ol>
0743
0744 <h4 id="algorithm-collect-known" algorithm>Collect `Credential`'s from the credential store</h4>
0745
0746 Given an [=environment settings object/origin=] (IoriginI) and
0747 a {{CredentialRequestOptions}} (IoptionsI), the user agent may
0748 <dfn abstract-op local-It="collect local">collect `Credential`'s from the credential store</dfn>,
0749 returning a set of {{Credential}} objects stored by the user agent locally that match IoptionsI'
0750 filter. If no such {{Credential}} objects are known, the returned set will be empty:
0751
0752 <ol class="algorithm">
0753 1. Let Ipossible matchesI be an empty set.
0754
0755 2. For each Iinterfacel in IoptionsI' <a>relevant credential interface objects</a>:
0756
0757   1. Let Irl be the result of executing Iinterfacel's
0758     {{Credential/[CollectFromCredentialStore]}(origin, options)}} internal method on
0759     IoriginI and IoptionsI.
0760
0761   2. If Irl is an [=exception=], return Irl.
0762
0763   3. Assert: Irl is a list of [=interface objects=].
0764
0765   4. For each Icl in Irl:
0766
0767     1. <a for="set">Append</a> Icl to Ipossible matchesI.
0768
0769 3. Return Ipossible matchesI.
```

```
0739 1. Let IcredentialsI be the result of <a abstract-op It="collect local">collecting
0740   `Credential`'s from the credential store</a>, given IoriginI and IoptionsI.
0741
0742 2. If IcredentialsI is an [=exception=], [=reject=] Ipl with IcredentialsI.
0743
0744 3. If all of the following statements are true, resolve Ipl with IcredentialsI[0] and
0745   skip the remaining steps:
0746
0747   1. IcredentialsI' [=set/size=] is 1
0748
0749   2. IoriginI does not [=origin/requires user mediation]require user mediation=]
0750
0751   3. IoptionsI is <a>matchable <i lang="la">a priori</i></a>.
0752
0753   4. IoptionsI.{{CredentialRequestOptions/mediation}} is not
0754     "{{CredentialMediationRequirement/required}}".
0755
0756   ISSUE: This might be the wrong model. It would be nice to support a site that wished
0757   to accept either username/passwords or webauthn-style credentials without forcing
0758   a chooser for those users who use the former, and who wish to remain signed in.
0759
0760 4. If IoptionsI' {{CredentialRequestOptions/mediation}} is
0761   "{{CredentialMediationRequirement/silent}}", [=resolve=] Ipl with `null`, and skip
0762   the remaining steps.
0763
0764 5. Let IchoiceI be the result of <a abstract-op It="ask to choose">asking the user to
0765   choose a `Credential` </a>, given IoptionsI and IcredentialsI.
0766
0767 6. If IchoiceI is `null` or a {{Credential}}, [=resolve=] Ipl with IchoiceI and skip the
0768   remaining steps.
0769
0770 7. Assert: IchoiceI is an [=interface object=].
0771
0772 8. Let IresultI be the result of executing IchoiceI's
0773   {{{DiscoverFromExternalSource}}}(origin, options)}, given IoriginI and IoptionsI.
0774
0775 9. If IresultI is a {{Credential}} or `null`, resolve Ipl with IresultI.
0776
0777   Otherwise, [=reject=] Ipl with IresultI.
0778
0779 7. Return Ipl.
0780 </ol>
0781
0782 <h4 id="algorithm-collect-known" algorithm>Collect `Credential`'s from the credential store</h4>
0783
0784 Given an [=environment settings object/origin=] (IoriginI) and
0785 a {{CredentialRequestOptions}} (IoptionsI), the user agent may
0786 <dfn abstract-op local-It="collect local">collect `Credential`'s from the credential store</dfn>,
0787 returning a set of {{Credential}} objects stored by the user agent locally that match IoptionsI'
0788 filter. If no such {{Credential}} objects are known, the returned set will be empty:
0789
0790 <ol class="algorithm">
0791 1. Let Ipossible matchesI be an empty set.
0792
0793 2. For each Iinterfacel in IoptionsI' <a>relevant credential interface objects</a>:
0794
0795   1. Let Irl be the result of executing Iinterfacel's
0796     {{Credential/[CollectFromCredentialStore]}(origin, options)}} internal method on
0797     IoriginI and IoptionsI.
0798
0799   2. If Irl is an [=exception=], return Irl.
0800
0801   3. Assert: Irl is a list of [=interface objects=].
0802
0803   4. For each Icl in Irl:
0804
0805     1. <a for="set">Append</a> Icl to Ipossible matchesI.
0806
0807 3. Return Ipossible matchesI.
```

```

0770 </ol>
0771
0772
0773 <h4 id="algorithm-store" algorithm>Store a `Credential` </h4>
0774
0775 The <dfn abstract-op>Store a `Credential` </dfn> algorithm accepts a {{Credential}}
0776 (Icredential), and returns a {{Promise}} which resolves once the object is persisted to the
0777 [=credential store=].
0778
0779 <ol class="algorithm">
0780 1. Let IsettingsI be the <a>current settings object</a>
0781
0782 2. Assert: IsettingsI is a [=secure context=].
0783
0784 3. Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
0785 are true:
0786
0787 1. IsettingsI does not have a [=environment settings object/responsible document=].
0788
0789 2. IsettingsI' [=environment settings object/responsible document=] is not the
0790 [=active document=] in a [=top-level browsing context=].
0791
0792 4. Let Ipl be [=a new promise=].
0793
0794 5. Run the following steps [=in parallel=]:
0795
0796 1. Let Irl be the result of executing Icredential's [=interface object=]'s
0797 {{Credential/[[Store]](credential)}} internal method on Icredential.
0798
0799 2. [=Resolve=] Ipl with Irl.
0800
0801 6. Return Ipl.
0802 </ol>
0803
0804 <h4 id="algorithm-create" algorithm>Create a `Credential` </h4>
0805
0806 The <dfn abstract-op>Create a `Credential` </dfn> algorithm accepts a {{CredentialCreationOptions}}
0807 (IoptionsI), and returns a {{Promise}} which resolves with a {{Credential}} if one can be created
0808 using the options provided, or `null` if no {{Credential}} can be created. In exceptional
0809 circumstances, the {{Promise}} may reject with an appropriate exception:
0810
0811 <ol class="algorithm">
0812 1. Let IsettingsI be the <a>current settings object</a>
0813
0814 2. Assert: IsettingsI is a [=secure context=].
0815
0816 3. Let IinterfacesI be the set of IoptionsI' <a>relevant credential interface objects</a>.
0817
0818 4. Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
0819
0820 are true:
0821
0822 1. IsettingsI does not have a [=environment settings object/responsible document=].
0823
0824 2. IsettingsI' [=environment settings object/responsible document=] is not the
0825 [=active document=] in a [=top-level browsing context=].
0826
0827 3. IinterfacesI' [=list/size=] is greater than 1.
0828
0829 Note: It may be reasonable at some point in the future to loosen this restriction, and
0830 allow the user agent to help the user choose among one of many potential credential
0831 types in order to support a "sign-up" use case. For the moment, though, we're punting
0832 on that by restricting the dictionary to a single entry.
0833
0834 5. Let Ipl be [=a new promise=].
0835
0836 6. Run the following steps [=in parallel=]:

```

```

0808 </ol>
0809
0810 <h4 id="algorithm-store" algorithm>Store a `Credential` </h4>
0811
0812 The <dfn abstract-op>Store a `Credential` </dfn> algorithm accepts a {{Credential}}
0813 (Icredential), and returns a {{Promise}} which resolves once the object is persisted to the
0814 [=credential store=].
0815
0816 <ol class="algorithm">
0817 1. Let IsettingsI be the <a>current settings object</a>
0818
0819 2. Assert: IsettingsI is a [=secure context=].
0820
0821 3. Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
0822 are true:
0823
0824 1. IsettingsI does not have a [=environment settings object/responsible document=].
0825
0826 2. IsettingsI' [=environment settings object/responsible document=] is not the
0827 [=active document=] in a [=top-level browsing context=].
0828
0829 4. Let Ipl be [=a new promise=].
0830
0831 5. Run the following steps [=in parallel=]:
0832
0833 1. Let Irl be the result of executing Icredential's [=interface object=]'s
0834 {{Credential/[[Store]](credential)}} internal method on Icredential.
0835
0836 2. [=Resolve=] Ipl with Irl.
0837
0838 6. Return Ipl.
0839 </ol>
0840
0841 <h4 id="algorithm-create" algorithm>Create a `Credential` </h4>
0842
0843 The <dfn abstract-op>Create a `Credential` </dfn> algorithm accepts a {{CredentialCreationOptions}}
0844 (IoptionsI), and returns a {{Promise}} which resolves with a {{Credential}} if one can be created
0845 using the options provided, or `null` if no {{Credential}} can be created. In exceptional
0846 circumstances, the {{Promise}} may reject with an appropriate exception:
0847
0848 <ol class="algorithm">
0849 1. Let IsettingsI be the <a>current settings object</a>
0850
0851 2. Assert: IsettingsI is a [=secure context=].
0852
0853 3. Let Iglobal be IsettingsI' [=environment settings object/global object=].
0854
0855 4. Let IinterfacesI be the [=set=] of IoptionsI' <a>relevant credential interface objects</a>.
0856
0857 5. Return [=a promise rejected with=] `NotSupportedError` if any of the following statements
0858 are true:
0859
0860 1. IsettingsI does not have a [=environment settings object/responsible document=].
0861
0862 2. IsettingsI' [=environment settings object/responsible document=] is not the
0863 [=active document=] in a [=top-level browsing context=].
0864
0865 3. IinterfacesI' [=list/size=] is greater than 1.
0866
0867 Note: It may be reasonable at some point in the future to loosen this restriction, and
0868 allow the user agent to help the user choose among one of many potential credential
0869 types in order to support a "sign-up" use case. For the moment, though, we're punting
0870 on that by restricting the dictionary to a single entry.
0871
0872 6. Let Ipl be [=a new promise=].
0873
0874 7. Let IoriginI be IsettingsI's [=environment settings object/origin=].
0875
0876

```

```

0837 1. Let lrl be the result of executing linterfaces[0] {{Credential/[[Create]](options)}}
0838     internal method on loptionsl.
0839
0840 2. If lrl is an [=exception=], [=reject=] lpl with lrl.
0841
0842     Otherwise, [=resolve=] lpl with lrl.
0843
0844 7. Return lpl.

```

```

0845 </ol>
0846
0847 <h4 id="algorithm-prevent-silent-access" algorithm>Prevent Silent Access</h4>
0848
0849 The <dfn abstract-op>Prevent Silent Access</dfn> algorithm accepts an [=environment settings
0850 object=] (lsettingsl), and returns a {{Promise}} which resolves once the `prevent silent access
0851 flag is persisted to the [=credential store=].
0852
0853 <ol class="algorithm">
0854 1. Let loriginl be lsettingsl' [=environment settings object/origin=].
0855
0856 2. Let lpl be [=a new promise=].
0857
0858 3. Run the following seps [=in parallel=]:
0859
0860     1. Set loriginl's <a>`prevent silent access` flag</a> in the [=credential store=].
0861
0862     2. [=Resolve=] lpl with `undefined`.
0863
0864 4. Return lpl.
0865 </ol>

```

```

0866 </section>
0867
0868 <!--
0869
0870
0871
0872
0873
0874
0875
0876 -->
0877 <section>
0878   # Password Credentials # {#passwords}
0879
0880   For good or for ill, many websites rely on username/password pairs as an authentication mechanism.
0881   The {{PasswordCredential}} interface is a [=credential=] meant to enable this use case, storing
0882   both a username and password, as well as metadata that can help a user choose the right account
0883   from within a [=credential chooser=].
0884
0885   ## Examples ## {#password-examples}
0886

```

```

0877 8. Run the following substeps [=in parallel=]:
0878
0879     1. Let lrl be the result of executing linterfaces[0]'s {{Credential/[[Create]](origin, options)}}
0880        internal method given loriginl and loptionsl.
0881
0882     2. If lrl is an [=exception=] or `null`, [=reject=] lpl with lrl, and terminate these substeps.
0883
0884     3. If lrl is a {{Credential}}, [=resolve=] lpl with lrl, and terminate these substeps.
0885
0886     4. Assert: lrl is a <code>[=constructCredential=]</code> algorithm.
0887
0888     5. [=Queue a task=] on lglobal's [=DOM manipulation task source=] to run the following substeps:
0889
0890         1. Let lresultl be the result of [=promise-calling=] lrl given lglobal.
0891         2. If lresultl is an [=exception=], [=reject=] lpl with lresultl and terminate these substeps.
0892         3. [=Resolve=] lpl with lresultl.
0893

```

```

0894 9. Return lpl.
0895 </ol>
0896
0897 <h4 id="algorithm-prevent-silent-access" algorithm>Prevent Silent Access</h4>
0898
0899 The <dfn abstract-op>Prevent Silent Access</dfn> algorithm accepts an [=environment settings
0900 object=] (lsettingsl), and returns a {{Promise}} which resolves once the `prevent silent access
0901 flag is persisted to the [=credential store=].
0902
0903 <ol class="algorithm">
0904 1. Let loriginl be lsettingsl' [=environment settings object/origin=].
0905
0906 2. Let lpl be [=a new promise=].
0907
0908 3. Run the following seps [=in parallel=]:
0909
0910     1. Set loriginl's <a>`prevent silent access` flag</a> in the [=credential store=].
0911
0912     2. [=Resolve=] lpl with `undefined`.
0913
0914 4. Return lpl.
0915 </ol>

```

```

0916 </section>
0917
0918 <!--
0919
0920
0921
0922
0923
0924
0925
0926
0927
0928 -->
0929 <section>
0930   # Password Credentials # {#passwords}
0931
0932   For good or for ill, many websites rely on username/password pairs as an authentication mechanism.
0933   The {{PasswordCredential}} interface is a [=credential=] meant to enable this use case, storing
0934   both a username and password, as well as metadata that can help a user choose the right account
0935   from within a [=credential chooser=].
0936
0937   ## Examples ## {#password-examples}
0938

```

```

0887   ### Password-based Sign-in ### {#examples-password-signin}
0888
0889   <div class="example">
0890     MegaCorp, Inc. supports passwords, and can use {{get()|navigator.credentials.get()}} to obtain
0891     username/password pairs from a user's [=credential store=]:
0892
0893     <pre>
0894     navigator.<a attribute>credentials</a>
0895     .<a idl it="get()" for="CredentialsContainer">get</a>({ '<a for="CredentialRequestOptions" dict-
0896     member>password</a>': true })
0897     .then(credential => {
0898       if (!credential) {
0899         // The user either doesn't have credentials for this site, or
0900         // refused to share them. Insert some code here to fall back to
0901         // a basic login form.
0902         return;
0903       }
0904       if (credential.<a attribute for="Credential">type</a> == '<a const>password</a>') {
0905         var form = new FormData();
0906         form.append('username_field', credential.id);
0907         form.append('password_field', credential.password);
0908         var opt = {
0909           method: 'POST',
0910           body: form,
0911           credentials: 'include' // Send cookies.
0912         };
0913         fetch('https://example.com/loginEndpoint', opt)
0914         .then(function (response) {
0915           if (/^ Iresponsel indicates a successful login */) {
0916             // Record that the credential was effective. See note below.
0917             navigator.<a attribute>credentials</a>.<a idl it="store()"
0918             for="CredentialsContainer">store</a>(credential);
0919             // Notify the user that sign-in succeeded! Do amazing, signed-in things!
0920             // Maybe navigate to a landing page via location.href =
0921             // '/signed-in-experience'?
0922           } else {
0923             // Insert some code here to fall back to a basic login form.
0924           }
0925         });
0926       }
0927     }
0928     </pre>
0929
0930     Alternatively, the website could just copy the credential data into a <{form}> and call
0931     {{HTMLFormElement/submit()}} on the form:
0932
0933     <pre>
0934     navigator.<a attribute>credentials</a>
0935     .<a idl it="get()" for="CredentialsContainer">get</a>({ '<a for="CredentialRequestOptions" dict-
0936     member>password</a>': true })
0937     .then(credential => {
0938       if (!credential) {
0939         return; // as above...
0940       }
0941       if (credential.<a attribute for="Credential">type</a> == '<a const href="#password-
0942       literal">password</a>') {
0943         document.querySelector('input[name=username_field]').value =
0944         credential.id;
0945         document.querySelector('input[name=password_field]').value =
0946         credential.password;
0947         document.querySelector('#myform').submit();
0948       }
0949     });
0950     </pre>
0951
0952     Note that the former method is much preferred, as it contains an explicit call
0953     to {{CredentialsContainer/store()}} and saves the credentials. The <{form}> based mechanism
0954     relies on form submission, which navigates the browsing context, making it difficult to
0955     ensure that {{store()}} is called after successful sign-in.

```

```

0939   ### Password-based Sign-in ### {#examples-password-signin}
0940
0941   <div class="example">
0942     MegaCorp, Inc. supports passwords, and can use {{get()|navigator.credentials.get()}} to obtain
0943     username/password pairs from a user's [=credential store=]:
0944
0945     <pre>
0946     navigator.<a attribute>credentials</a>
0947     .<a idl it="get()" for="CredentialsContainer">get</a>({ '<a for="CredentialRequestOptions" dict-
0948     member>password</a>': true })
0949     .then(credential => {
0950       if (!credential) {
0951         // The user either doesn't have credentials for this site, or
0952         // refused to share them. Insert some code here to fall back to
0953         // a basic login form.
0954         return;
0955       }
0956       if (credential.<a attribute for="Credential">type</a> == '<a const>password</a>') {
0957         var form = new FormData();
0958         form.append('username_field', credential.id);
0959         form.append('password_field', credential.password);
0960         var opt = {
0961           method: 'POST',
0962           body: form,
0963           credentials: 'include' // Send cookies.
0964         };
0965         fetch('https://example.com/loginEndpoint', opt)
0966         .then(function (response) {
0967           if (/^ Iresponsel indicates a successful login */) {
0968             // Record that the credential was effective. See note below.
0969             navigator.<a attribute>credentials</a>.<a idl it="store()"
0970             for="CredentialsContainer">store</a>(credential);
0971             // Notify the user that sign-in succeeded! Do amazing, signed-in things!
0972             // Maybe navigate to a landing page via location.href =
0973             // '/signed-in-experience'?
0974           } else {
0975             // Insert some code here to fall back to a basic login form.
0976           }
0977         });
0978       }
0979     }
0980     </pre>
0981
0982     Alternatively, the website could just copy the credential data into a <{form}> and call
0983     {{HTMLFormElement/submit()}} on the form:
0984
0985     <pre>
0986     navigator.<a attribute>credentials</a>
0987     .<a idl it="get()" for="CredentialsContainer">get</a>({ '<a for="CredentialRequestOptions" dict-
0988     member>password</a>': true })
0989     .then(credential => {
0990       if (!credential) {
0991         return; // as above...
0992       }
0993       if (credential.<a attribute for="Credential">type</a> == '<a const href="#password-
0994       literal">password</a>') {
0995         document.querySelector('input[name=username_field]').value =
0996         credential.id;
0997         document.querySelector('input[name=password_field]').value =
0998         credential.password;
0999         document.querySelector('#myform').submit();
1000       }
1001     });
1002     </pre>
1003
1004     Note that the former method is much preferred, as it contains an explicit call
1005     to {{CredentialsContainer/store()}} and saves the credentials. The <{form}> based mechanism
1006     relies on form submission, which navigates the browsing context, making it difficult to
1007     ensure that {{store()}} is called after successful sign-in.

```

```

0952 </div>
0953
0954 Note: The [=credential chooser=] presented by the user agent could allow the user to choose
0955 credentials that aren't actually stored for the current origin. For instance, it might offer up
0956 credentials from `https://m.example.com` when signing into `https://www.example.com` (as
0957 described in [[#security-credential-access]], or it might allow a user to create a new
0958 credential on the fly. Developers can deal gracefully with this uncertainty by calling
0959 {{CredentialsContainer/store()}} every time credentials are successfully used, even right after
0960 credentials have been retrieved from {{CredentialsContainer/get()}}: if the credentials aren't
0961 yet stored for the origin, the user will be given the opportunity to do so. If they are stored,
0962 the user won't be prompted.
0963
0964 ### Post-sign-in Confirmation ### {#examples-post-signin}
0965
0966 To ensure that users are offered to store new credentials after a successful sign-in, they can
0967 to be passed to {{CredentialsContainer/store()}}.
0968
0969 <div class="example">
0970   If a user is signed in by submitting the credentials to a sign-in endpoint via {{fetch()}},
0971   we can check the response to determine whether the user
0972   was signed in successfully, and notify the user agent accordingly. Given a sign-in form like the
0973   following:
0974
0975   <pre>
0976     &lt;form action="https://example.com/login" method="POST" id="theForm"&gt;
0977       &lt;label for="username"&gt;Username&lt;/label&gt;
0978       &lt;input type="text" id="username" name="username" <a element-attr
0979 for="input">autocomplete</a>="<a attr-value>username</a>"&gt;
0980       &lt;label for="password"&gt;Password&lt;/label&gt;
0981       &lt;input type="password" id="password" name="password" <a element-attr
0982 for="input">autocomplete</a>="<a attr-value>current-password</a>"&gt;
0983       &lt;input type="submit"&gt;
0984     &lt;/form&gt;
0985   </pre>
0986
0987   Then the developer can handle the form submission with something like the following handler:
0988
0989   <pre>
0990     document.querySelector('#theForm').addEventListener('submit', e =&gt; {
0991       if (<a attribute lt="credentials">navigator.credentials</a> {
0992         e.preventDefault();
0993
0994         // Construct a new <a id>PasswordCredential</a> from the <a id>HTMLFormElement</a>
0995         // that fired the "submit" event: this will suck up the values of the fields
0996         // labeled with "username" and "current-password" <a element-attr
0997 for="input">autocomplete</a>
0998         // attributes:
0999         var c = new <a idl lt="PasswordCredential(form)">PasswordCredential</a>(e.target);
1000
1001         // Fetch the form's action URL, passing that new credential object in
1002         // as a FormData object. If the response indicates success, tell the user agent
1003         // so it can ask the user to store the password for future use:
1004         var opt = {
1005           method: 'POST',
1006           body: new FormData(e.target),
1007           credentials: 'include' // Send cookies.
1008         };
1009         fetch(e.target.action, opt).then(r =&gt; {
1010           if (/^ Irl is a "successful" <a id>Response</a> */)
1011             <a idl lt="store()">navigator.credentials.store</a>(c);
1012         });
1013       }
1014     });
1015   </pre>
1016 </div>
1017
1018 ### Change Password ### {#examples-change-password}
1019
1020 This same storage mechanism can be reused for "password change" with no modifications: if the

```

```

1004 </div>
1005
1006 Note: The [=credential chooser=] presented by the user agent could allow the user to choose
1007 credentials that aren't actually stored for the current origin. For instance, it might offer up
1008 credentials from `https://m.example.com` when signing into `https://www.example.com` (as
1009 described in [[#security-credential-access]], or it might allow a user to create a new
1010 credential on the fly. Developers can deal gracefully with this uncertainty by calling
1011 {{CredentialsContainer/store()}} every time credentials are successfully used, even right after
1012 credentials have been retrieved from {{CredentialsContainer/get()}}: if the credentials aren't
1013 yet stored for the origin, the user will be given the opportunity to do so. If they are stored,
1014 the user won't be prompted.
1015
1016 ### Post-sign-in Confirmation ### {#examples-post-signin}
1017
1018 To ensure that users are offered to store new credentials after a successful sign-in, they can
1019 to be passed to {{CredentialsContainer/store()}}.
1020
1021 <div class="example">
1022   If a user is signed in by submitting the credentials to a sign-in endpoint via {{fetch()}},
1023   we can check the response to determine whether the user
1024   was signed in successfully, and notify the user agent accordingly. Given a sign-in form like the
1025   following:
1026
1027   <pre>
1028     &lt;form action="https://example.com/login" method="POST" id="theForm"&gt;
1029       &lt;label for="username"&gt;Username&lt;/label&gt;
1030       &lt;input type="text" id="username" name="username" <a element-attr
1031 for="input">autocomplete</a>="<a attr-value>username</a>"&gt;
1032       &lt;label for="password"&gt;Password&lt;/label&gt;
1033       &lt;input type="password" id="password" name="password" <a element-attr
1034 for="input">autocomplete</a>="<a attr-value>current-password</a>"&gt;
1035       &lt;input type="submit"&gt;
1036     &lt;/form&gt;
1037   </pre>
1038
1039   Then the developer can handle the form submission with something like the following handler:
1040
1041   <pre>
1042     document.querySelector('#theForm').addEventListener('submit', e =&gt; {
1043       if (<a attribute lt="credentials">navigator.credentials</a> {
1044         e.preventDefault();
1045
1046         // Construct a new <a id>PasswordCredential</a> from the <a id>HTMLFormElement</a>
1047         // that fired the "submit" event: this will suck up the values of the fields
1048         // labeled with "username" and "current-password" <a element-attr
1049 for="input">autocomplete</a>
1050         // attributes:
1051         var c = new <a idl lt="PasswordCredential(form)">PasswordCredential</a>(e.target);
1052
1053         // Fetch the form's action URL, passing that new credential object in
1054         // as a FormData object. If the response indicates success, tell the user agent
1055         // so it can ask the user to store the password for future use:
1056         var opt = {
1057           method: 'POST',
1058           body: new FormData(e.target),
1059           credentials: 'include' // Send cookies.
1060         };
1061         fetch(e.target.action, opt).then(r =&gt; {
1062           if (/^ Irl is a "successful" <a id>Response</a> */)
1063             <a idl lt="store()">navigator.credentials.store</a>(c);
1064         });
1065       }
1066     });
1067   </pre>
1068 </div>
1069
1070 ### Change Password ### {#examples-change-password}
1071
1072 This same storage mechanism can be reused for "password change" with no modifications: if the

```

1018 user changes their credentials, the website can notify the user agent that they've successfully
1019 signed in with new credentials. The user agent can then update the credentials it stores:
1020
1021 <div class="example">
1022 MegaCorp Inc. allows users to change their passwords by POSTing data to
1023 a backend server asynchronously. After doing so successfully, they can
1024 update the user's credentials by calling {{CredentialsContainer/store()}}
1025 with the new information.
1026
1027 Given a password change form like the following:
1028
1029 <pre>
1030 <form action="https://example.com/changePassword" method="POST" id="theForm">
1031 <input type="hidden" name="username" <a element-attr for="input">autocomplete=<a attr-
1032 value>username" value="user">
1033 <label for="password">New Password</label>
1034 <input type="password" id="password" name="password" <a element-attr
1035 for="input">autocomplete=<a attr-value>new-password">
1036 <input type="submit">
1037 </form>
1038 </pre>
1039
1040 The developer can handle the form submission with something like the following:
1041
1042 <pre>
1043 document.querySelector('#theForm').addEventListener('submit', e => {
1044 if (<a attribute lt="credentials">navigator.credentials {
1045 e.preventDefault();
1046
1047 // Construct a new <a idl>PasswordCredential from the <a idl>HTMLFormElement
1048 // that fired the "submit" event: this will suck up the values of the fields
1049 // labeled with "username" and "new-password" <a element-attr for="input">autocomplete
1050 // attributes:
1051 var c = new <a idl lt="PasswordCredential(form)">PasswordCredential(e.target);
1052
1053 // Fetch the form's action URL, passing that new credential object in
1054 // as a FormData object. If the response indicates success, tell the user agent
1055 // so it can ask the user to store the password for future use:
1056 var opt = {
1057 method: 'POST',
1058 body: new FormData(e.target),
1059 credentials: 'include' // Send cookies.
1060 };
1061 fetch(e.target.action, opt).then(r => {
1062 if (!r.is a "successful" <a idl>Response *)
1063 <a idl lt="store()">navigator.credentials.store(c);
1064 });
1065 }
1066 }
1067 </pre>
1068 </div>
1069
1070 ## The `PasswordCredential` Interface ## {#passwordcredential-interface}
1071
1072 <pre class="idl">
1073 typedef (FormData or URLSearchParams) CredentialBodyType;
1074
1075 [Constructor(HTMLFormElement form),
1076 Constructor(PasswordCredentialData data),
1077 Exposed=Window,
1078 SecureContext]
1079 interface PasswordCredential : Credential {
1080 readonly attribute USVString password;
1081 };
1082 PasswordCredential implements CredentialUserData;
1083
1084 partial dictionary CredentialRequestOptions {
1085 boolean password = false;
1086 };

1070 user changes their credentials, the website can notify the user agent that they've successfully
1071 signed in with new credentials. The user agent can then update the credentials it stores:
1072
1073 <div class="example">
1074 MegaCorp Inc. allows users to change their passwords by POSTing data to
1075 a backend server asynchronously. After doing so successfully, they can
1076 update the user's credentials by calling {{CredentialsContainer/store()}}
1077 with the new information.
1078
1079 Given a password change form like the following:
1080
1081 <pre>
1082 <form action="https://example.com/changePassword" method="POST" id="theForm">
1083 <input type="hidden" name="username" <a element-attr for="input">autocomplete=<a attr-
1084 value>username" value="user">
1085 <label for="password">New Password</label>
1086 <input type="password" id="password" name="password" <a element-attr
1087 for="input">autocomplete=<a attr-value>new-password">
1088 <input type="submit">
1089 </form>
1090 </pre>
1091
1092 The developer can handle the form submission with something like the following:
1093
1094 <pre>
1095 document.querySelector('#theForm').addEventListener('submit', e => {
1096 if (<a attribute lt="credentials">navigator.credentials {
1097 e.preventDefault();
1098
1099 // Construct a new <a idl>PasswordCredential from the <a idl>HTMLFormElement
1100 // that fired the "submit" event: this will suck up the values of the fields
1101 // labeled with "username" and "new-password" <a element-attr for="input">autocomplete
1102 // attributes:
1103 var c = new <a idl lt="PasswordCredential(form)">PasswordCredential(e.target);
1104
1105 // Fetch the form's action URL, passing that new credential object in
1106 // as a FormData object. If the response indicates success, tell the user agent
1107 // so it can ask the user to store the password for future use:
1108 var opt = {
1109 method: 'POST',
1110 body: new FormData(e.target),
1111 credentials: 'include' // Send cookies.
1112 };
1113 fetch(e.target.action, opt).then(r => {
1114 if (!r.is a "successful" <a idl>Response *)
1115 <a idl lt="store()">navigator.credentials.store(c);
1116 });
1117 }
1118 }
1119 </pre>
1120 </div>
1121
1122 ## The `PasswordCredential` Interface ## {#passwordcredential-interface}
1123
1124 <pre class="idl">
1125 typedef (FormData or URLSearchParams) CredentialBodyType;
1126
1127 [Constructor(HTMLFormElement form),
1128 Constructor(PasswordCredentialData data),
1129 Exposed=Window,
1130 SecureContext]
1131 interface PasswordCredential : Credential {
1132 readonly attribute USVString password;
1133 };
1134 PasswordCredential implements CredentialUserData;
1135
1136 partial dictionary CredentialRequestOptions {
1137 boolean password = false;
1138 };

```
1085 </pre>
1086 <dl dfn-for="PasswordCredential">
1087 : <dfn attribute>password</dfn>
1088 :: This attribute represents the password of the credential.
1089
1090 : {{Credential/[[type]]}}
1091 :: The {{PasswordCredential}} [=interface object=] has an internal slot named `[[type]]` whose
1092 value is "<dfn const for="Credential/[[type]]"> password` </dfn>".
1093
1094 : {{Credential/[[discovery]]}}
1095 :: The {{PasswordCredential}} [=interface object=] has an internal slot named `[[discovery]]`
1096 whose value is "{{Credential/[[discovery]]/credential store}}".
1097
1098 : <dfn constructor>PasswordCredential(form)</dfn>
1099 :: This constructor accepts an {{HTMLFormElement}} (lforml), and runs the following steps:
1100
1101 1. Let lrl be the result of executing <a abstract-op>Create a `PasswordCredential` from
1102 an `HTMLFormElement` </a> on lforml.
1103
1104 2. If lrl is an [=exception=], [=throw=] lrl.
1105
1106 Otherwise, return lrl.
1107
1108 : <dfn constructor>PasswordCredential(data)</dfn>
1109 :: This constructor accepts a {{PasswordCredentialData}} (ldata), and runs the following steps:
1110
1111 1. Let lrl be the result of executing <a abstract-op>Create a `PasswordCredential` from
1112 `PasswordCredentialData` </a> on ldata.
1113
1114 2. If lrl is an [=exception=], [=throw=] lrl.
1115
1116 Otherwise, return lrl.
1117 </dl>
1118
1119 {{PasswordCredential}} objects can be created via
1120 {{CredentialsContainer/create()/navigator.credentials.create()}}
1121 either explicitly by passing in a {{PasswordCredentialData}} dictionary, or based on the contents
1122 of an {{HTMLFormElement}}'s [=submittable elements=].
1123
1124 <pre class="idl">
1125 dictionary PasswordCredentialData : CredentialData {
1126   USVString name;
1127   USVString iconURL;
1128   required USVString password;
1129 };
1130
1131 typedef (PasswordCredentialData or HTMLFormElement) PasswordCredentialInit;
1132
1133 partial dictionary CredentialCreationOptions {
1134   PasswordCredentialInit password;
1135 };
1136 </pre>
1137
1138 {{PasswordCredential}} objects are [=Credential/origin bound=].
1139
1140 {{PasswordCredential}}'s [=interface object=] inherits {{Credential}}'s implementation of
1141 {{Credential/[[DiscoverFromExternalSource]](origin, options)}}, and defines its own implementation of
1142 {{PasswordCredential/[[CollectFromCredentialStore]](origin, options)}},
1143 {{PasswordCredential/[[Create]](options)}}, and
1144 {{PasswordCredential/[[Store]](credential)}}.
1145
1146 ## Algorithms ## {#passwordcredential-algorithms}
1147
1148 <h4 algorithm id="collectfromcredentialstore-passwordcredential">
1149 `PasswordCredential`'s `[[CollectFromCredentialStore]](origin, options)`
1150 </h4>
1151
1152 <dfn for="PasswordCredential" method>[[CollectFromCredentialStore]](origin, options)</dfn> is
called
```

```
1137 </pre>
1138 <dl dfn-for="PasswordCredential">
1139 : <dfn attribute>password</dfn>
1140 :: This attribute represents the password of the credential.
1141
1142 : {{Credential/[[type]]}}
1143 :: The {{PasswordCredential}} [=interface object=] has an internal slot named `[[type]]` whose
1144 value is "<dfn const for="Credential/[[type]]"> password` </dfn>".
1145
1146 : {{Credential/[[discovery]]}}
1147 :: The {{PasswordCredential}} [=interface object=] has an internal slot named `[[discovery]]`
1148 whose value is "{{Credential/[[discovery]]/credential store}}".
1149
1150 : <dfn constructor>PasswordCredential(form)</dfn>
1151 :: This constructor accepts an {{HTMLFormElement}} (lforml), and runs the following steps:
1152
1153 1. Let lrl be the result of executing <a abstract-op>Create a `PasswordCredential` from
1154 an `HTMLFormElement` </a> on lforml.
1155
1156 2. If lrl is an [=exception=], [=throw=] lrl.
1157
1158 Otherwise, return lrl.
1159
1160 : <dfn constructor>PasswordCredential(data)</dfn>
1161 :: This constructor accepts a {{PasswordCredentialData}} (ldata), and runs the following steps:
1162
1163 1. Let lrl be the result of executing <a abstract-op>Create a `PasswordCredential` from
1164 `PasswordCredentialData` </a> on ldata.
1165
1166 2. If lrl is an [=exception=], [=throw=] lrl.
1167
1168 Otherwise, return lrl.
1169 </dl>
1170
1171 {{PasswordCredential}} objects can be created via
1172 {{CredentialsContainer/create()/navigator.credentials.create()}}
1173 either explicitly by passing in a {{PasswordCredentialData}} dictionary, or based on the contents
1174 of an {{HTMLFormElement}}'s [=submittable elements=].
1175
1176 <pre class="idl">
1177 dictionary PasswordCredentialData : CredentialData {
1178   USVString name;
1179   USVString iconURL;
1180   required USVString password;
1181 };
1182
1183 typedef (PasswordCredentialData or HTMLFormElement) PasswordCredentialInit;
1184
1185 partial dictionary CredentialCreationOptions {
1186   PasswordCredentialInit password;
1187 };
1188 </pre>
1189
1190 {{PasswordCredential}} objects are [=Credential/origin bound=].
1191
1192 {{PasswordCredential}}'s [=interface object=] inherits {{Credential}}'s implementation of
1193 {{Credential/[[DiscoverFromExternalSource]](origin, options)}}, and defines its own implementation of
1194 {{PasswordCredential/[[CollectFromCredentialStore]](origin, options)}},
1195 {{PasswordCredential/[[Create]](origin, options)}}, and
1196 {{PasswordCredential/[[Store]](credential)}}.
1197
1198 ## Algorithms ## {#passwordcredential-algorithms}
1199
1200 <h4 algorithm id="collectfromcredentialstore-passwordcredential">
1201 `PasswordCredential`'s `[[CollectFromCredentialStore]](origin, options)`
1202 </h4>
1203
1204 <dfn for="PasswordCredential" method>[[CollectFromCredentialStore]](origin, options)</dfn> is
called
```

```
1153 with an [=environment settings object/origin=] (loriginl) and a {{CredentialRequestOptions}} (l
1154 optionsl),
1155 and returns a set of {{Credential}} objects from
1156 the [=credential store=]. If no matching {{Credential}} objects are available, the returned set
1157 will be empty.
1158 <ol class="algorithm">
1159 1. Assert: loptionsl["{{CredentialRequestOptions/password}}"] [=map/exists=].
1160
1161 2. If loptionsl["{{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 loriginl.
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}} (loptionsl), 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
1178 be
1179 used to create a {{PasswordCredential}}, this algorithm will return a {{TypeError}} [=exception=].
1180
1181 <ol class="algorithm">
1182 1. Assert: loptionsl["{{CredentialCreationOptions/password}}"] [=map/exists=].
1183
1184 2. If loptionsl["{{CredentialCreationOptions/password}}"] is an {{HTMLFormElement}}, return the
1185 result of executing <a abstract-op>Create a `PasswordCredential` from an
1186 `HTMLFormElement` </a> on loptionsl["{{CredentialCreationOptions/password}}"].
1187
1188 3. If loptionsl["{{CredentialCreationOptions/password}}"] is a {{PasswordCredentialData}}, return
1189 the result of executing <a abstract-op>Create a `PasswordCredential` from
1190 `PasswordCredentialData` </a> on loptionsl["{{CredentialCreationOptions/password}}"].
1191
1192 4. Return a {{TypeError}} [=exception=].
1193 </ol>
1194
1195 <h4 algorithm id="store-passwordcredential">
1196 `PasswordCredential`'s [[Store]](credential)
1197 </h4>
1198
1199 <dfn for="PasswordCredential" method>[[Store]](credential)</dfn> is called with a
1200 {{PasswordCredential}} (lcredentiall), and returns once lcredentiall is persisted to the
1201 [=credential store=].
1202
1203 <ol class="algorithm">
1204 1. If the user agent's [=credential store=] contains a {{PasswordCredential}} (lstoredl)
1205 whose {{Credential/id}} attribute is lcredentiall's {{Credential/id}} and whose
1206 {{[[origin]]}} slot is the [=same origin=] as lcredentiall's {{Credential/[[origin]]}},
1207 then:
1208
1209 1. If the user grants permission to update credentials (as discussed when defining
1210 [=user mediation=]), then:
1211
1212 1. Set lstoredl's <a attribute for="PasswordCredential">`password` </a> to lcredentiall's
1213 <a attribute for="PasswordCredential">`password` </a>.
1214
1215 2. Set lstoredl's {{CredentialUserData/name}} to lcredentiall's
1216 {{CredentialUserData/name}}.
1217
1218 3. Set lstoredl's {{CredentialUserData/iconURL}} to lcredentiall's
1219 {{CredentialUserData/iconURL}}.
```

```
1205 with an [=environment settings object/origin=] (loriginl) and a {{CredentialRequestOptions}} (l
1206 optionsl),
1207 and returns a set of {{Credential}} objects from
1208 the [=credential store=]. If no matching {{Credential}} objects are available, the returned set
1209 will be empty.
1210 <ol class="algorithm">
1211 1. Assert: loptionsl["{{CredentialRequestOptions/password}}"] [=map/exists=].
1212
1213 2. If loptionsl["{{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 loriginl.
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}} (loptionsl), 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
1231 be
1232 used to create a {{PasswordCredential}}, this algorithm will return a {{TypeError}} [=exception=].
1233
1234 <ol class="algorithm">
1235 1. Assert: loptionsl["{{CredentialCreationOptions/password}}"] [=map/exists=].
1236
1237 2. If loptionsl["{{CredentialCreationOptions/password}}"] is an {{HTMLFormElement}}, return the
1238 result of executing <a abstract-op>Create a `PasswordCredential` from an
1239 `HTMLFormElement` </a> on loptionsl["{{CredentialCreationOptions/password}}"].
1240
1241 3. If loptionsl["{{CredentialCreationOptions/password}}"] is a {{PasswordCredentialData}}, return
1242 the result of executing <a abstract-op>Create a `PasswordCredential` from
1243 `PasswordCredentialData` </a> on loptionsl["{{CredentialCreationOptions/password}}"].
1244
1245 4. Return a {{TypeError}} [=exception=].
1246 </ol>
1247
1248 <h4 algorithm id="store-passwordcredential">
1249 `PasswordCredential`'s [[Store]](credential)
1250 </h4>
1251
1252 <dfn for="PasswordCredential" method>[[Store]](credential)</dfn> is called with a
1253 {{PasswordCredential}} (lcredentiall), and returns once lcredentiall is persisted to the
1254 [=credential store=].
1255
1256 <ol class="algorithm">
1257 1. If the user agent's [=credential store=] contains a {{PasswordCredential}} (lstoredl)
1258 whose {{Credential/id}} attribute is lcredentiall's {{Credential/id}} and whose
1259 {{[[origin]]}} slot is the [=same origin=] as lcredentiall's {{Credential/[[origin]]}},
1260 then:
1261
1262 1. If the user grants permission to update credentials (as discussed when defining
1263 [=user mediation=]), then:
1264
1265 1. Set lstoredl's <a attribute for="PasswordCredential">`password` </a> to lcredentiall's
1266 <a attribute for="PasswordCredential">`password` </a>.
1267
1268 2. Set lstoredl's {{CredentialUserData/name}} to lcredentiall's
1269 {{CredentialUserData/name}}.
1270
1271 3. Set lstoredl's {{CredentialUserData/iconURL}} to lcredentiall's
1272 {{CredentialUserData/iconURL}}.
```

```

1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285

```

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

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

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

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

```

<ol class="algorithm">
  1. Let ldata be a new {{PasswordCredentialData}} dictionary.
  2. Let lformData be the result of executing the {{FormData}} constructor on lforml.
  3. Let lelements be a list of all the [=submittable elements=] whose [=form owner=] is lforml, in [=tree order=].
  4. Let lnewPasswordObserved be `false`.
  5. For each lfield in lelements, run the following steps:
    1. If lfield does not have an <{input/autocomplete}> attribute, then skip to the next lfield.
    2. Let lname be the value of lfield's <{input/name}> attribute.
    3. If lformData's {{FormData/has()}} method returns `false` when executed on lname, then skip to the next lfield.
    4. If lfield's <{input/autocomplete}> attribute's value contains one or more [=autofill detail tokens=] (ltokens), then:
      1. For each ltoken in ltokens:
        1. If ltoken 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 ldata's {{PasswordCredentialData/password}} member's value to the result of executing lformData's {{FormData/get()}} method on lname, and lnewPasswordObserved to `true`.
          : "<a attr-value>`current-password`</a>"
          :: If lnewPasswordObserved is `false`, set ldata's {{PasswordCredentialData/password}} member's value to the result of executing lformData's

```

```

1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338

```

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

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

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

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

```

<ol class="algorithm">
  1. Let ldata be a new {{PasswordCredentialData}} dictionary.
  2. Let lformData be the result of executing the {{FormData}} constructor on lforml.
  3. Let lelements be a list of all the [=submittable elements=] whose [=form owner=] is lforml, in [=tree order=].
  4. Let lnewPasswordObserved be `false`.
  5. For each lfield in lelements, run the following steps:
    1. If lfield does not have an <{input/autocomplete}> attribute, then skip to the next lfield.
    2. Let lname be the value of lfield's <{input/name}> attribute.
    3. If lformData's {{FormData/has()}} method returns `false` when executed on lname, then skip to the next lfield.
    4. If lfield's <{input/autocomplete}> attribute's value contains one or more [=autofill detail tokens=] (ltokens), then:
      1. For each ltoken in ltokens:
        1. If ltoken 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 ldata's {{PasswordCredentialData/password}} member's value to the result of executing lformData's {{FormData/get()}} method on lname, and lnewPasswordObserved to `true`.
          : "<a attr-value>`current-password`</a>"
          :: If lnewPasswordObserved is `false`, set ldata's {{PasswordCredentialData/password}} member's value to the result of executing lformData's

```

```
1286 {{FormData/get()}} method on Inamel.  
1287  
1288 Note: By checking that InewPasswordObservedl is `false`,  
1289 `new-password` fields take precedence over  
1290 `current-password` fields.  
1291  
1292 : "<a attr-value>`photo` </a>"  
1293 :: Set ldata's {{CredentialUserData/iconURL}} member's  
1294 value to the result of executing lformData's  
1295 {{FormData/get()}} method on Inamel.  
1296  
1297 : "<a attr-value>`name` </a>"  
1298 : "<a attr-value>`nickname` </a>"  
1299 :: Set ldata's {{CredentialUserData/name}} member's  
1300 value to the result of executing lformData's  
1301 {{FormData/get()}} method on Inamel.  
1302  
1303 : "<a attr-value>`username` </a>"  
1304 :: Set ldata's {{CredentialData/id}} member's value to the  
1305 result of executing lformData's {{FormData/get()}} method  
1306 on Inamel.  
1307  
1308 6. Let lcl be the result of executing <a abstract-op>Create a `PasswordCredential` from  
1309 `PasswordCredentialData` </a> on ldata.  
1310  
1311 7. If lcl is an [=exception=], return lcl.  
1312  
1313 8. Assert: lcl is a {{PasswordCredential}}.  
1314  
1315 9. Return lcl.  
1316 </ol>  
1317  
1318 <h4 algorithm id="construct-passwordcredential-data">  
1319 Create a `PasswordCredential` from `PasswordCredentialData`  
1320 </h4>  
1321  
1322 To <dfn abstract-op>Create a `PasswordCredential` from `PasswordCredentialData` </dfn>, given an  
1323 {{PasswordCredentialData}} (ldata), run these steps.  
1324  
1325 <ol class="algorithm">  
1326 1. Let lcl be a new {{PasswordCredential}} object.  
1327  
1328 2. If any of the following are the empty string, return a {{TypeError}} [=exception=]:  
1329  
1330 * ldata's {{CredentialData/id}} member's value  
1331 * ldata's {{PasswordCredentialData/password}} member's value  
1332  
1333 3. Set lcl's properties as follows:  
1334  
1335 : <a attribute for="PasswordCredential">`password` </a>  
1336 :: ldata's {{PasswordCredentialData/password}} member's value  
1337 : {{Credential/id}}  
1338 :: ldata's {{CredentialData/id}} member's value  
1339 : {{CredentialUserData/iconURL}}  
1340 :: ldata's {{PasswordCredentialData/iconURL}} member's value  
1341 : {{CredentialUserData/name}}  
1342 :: ldata's {{PasswordCredentialData/name}} member's value  
1343 : {{Credential/[[origin]]}}  
1344 :: The [=environment settings object/origin=] of the [=current settings object=].  
1345  
1346 4. Return lcl.  
1347 </ol>  
1348  
1349 <h4 algorithm id="passwordcredential-matching">  
1350 `CredentialRequestOptions` Matching for `PasswordCredential`  
1351 </h4>  
1352  
1353 Given a {{CredentialRequestOptions}} (loptionsl), the following algorithm returns "" Matches`" if  
1354 the {{PasswordCredential}} should be available as a response to a {{CredentialsContainer/get()}}
```

```
1339 {{FormData/get()}} method on Inamel.  
1340  
1341 Note: By checking that InewPasswordObservedl is `false`,  
1342 `new-password` fields take precedence over  
1343 `current-password` fields.  
1344  
1345 : "<a attr-value>`photo` </a>"  
1346 :: Set ldata's {{CredentialUserData/iconURL}} member's  
1347 value to the result of executing lformData's  
1348 {{FormData/get()}} method on Inamel.  
1349  
1350 : "<a attr-value>`name` </a>"  
1351 : "<a attr-value>`nickname` </a>"  
1352 :: Set ldata's {{CredentialUserData/name}} member's  
1353 value to the result of executing lformData's  
1354 {{FormData/get()}} method on Inamel.  
1355  
1356 : "<a attr-value>`username` </a>"  
1357 :: Set ldata's {{CredentialData/id}} member's value to the  
1358 result of executing lformData's {{FormData/get()}} method  
1359 on Inamel.  
1360  
1361 6. Let lcl be the result of executing <a abstract-op>Create a `PasswordCredential` from  
1362 `PasswordCredentialData` </a> on ldata.  
1363  
1364 7. If lcl is an [=exception=], return lcl.  
1365  
1366 8. Assert: lcl is a {{PasswordCredential}}.  
1367  
1368 9. Return lcl.  
1369 </ol>  
1370  
1371 <h4 algorithm id="construct-passwordcredential-data">  
1372 Create a `PasswordCredential` from `PasswordCredentialData`  
1373 </h4>  
1374  
1375 To <dfn abstract-op>Create a `PasswordCredential` from `PasswordCredentialData` </dfn>, given an  
1376 {{PasswordCredentialData}} (ldata), run these steps.  
1377  
1378 <ol class="algorithm">  
1379 1. Let lcl be a new {{PasswordCredential}} object.  
1380  
1381 2. If any of the following are the empty string, return a {{TypeError}} [=exception=]:  
1382  
1383 * ldata's {{CredentialData/id}} member's value  
1384 * ldata's {{PasswordCredentialData/password}} member's value  
1385  
1386 3. Set lcl's properties as follows:  
1387  
1388 : <a attribute for="PasswordCredential">`password` </a>  
1389 :: ldata's {{PasswordCredentialData/password}} member's value  
1390 : {{Credential/id}}  
1391 :: ldata's {{CredentialData/id}} member's value  
1392 : {{CredentialUserData/iconURL}}  
1393 :: ldata's {{PasswordCredentialData/iconURL}} member's value  
1394 : {{CredentialUserData/name}}  
1395 :: ldata's {{PasswordCredentialData/name}} member's value  
1396 : {{Credential/[[origin]]}}  
1397 :: The [=environment settings object/origin=] of the [=current settings object=].  
1398  
1399 4. Return lcl.  
1400 </ol>  
1401  
1402 <h4 algorithm id="passwordcredential-matching">  
1403 `CredentialRequestOptions` Matching for `PasswordCredential`  
1404 </h4>  
1405  
1406 Given a {{CredentialRequestOptions}} (loptionsl), the following algorithm returns "" Matches`" if  
1407 the {{PasswordCredential}} should be available as a response to a {{CredentialsContainer/get()}}
```

```
1355 request, and "Does Not Match" otherwise.
1356
1357 1. If options has a  CredentialRequestOptions/password member whose value is true, then
1358 return "Matches".
1359
1360 2. Return "Does Not Match".
1361
1362 </section>
1363
1364 <!--
1365
1366
1367
1368
1369
1370
1371
1372 -->
1373 <section>
1374 # Federated Credentials # {#federated}
1375
1376 ## The `FederatedCredential` Interface ## {#federatedcredential-interface}
1377
1378 <pre class="idl">
1379 [Constructor(FederatedCredentialInit data),
1380 Exposed=Window,
1381 SecureContext]
1382 interface FederatedCredential : Credential {
1383   readonly attribute USVString provider;
1384   readonly attribute DOMString? protocol;
1385 };
1386 FederatedCredential implements CredentialUserData;
1387
1388 dictionary FederatedCredentialRequestOptions {
1389   sequence<USVString> providers;
1390   sequence<DOMString> protocols;
1391 };
1392
1393 partial dictionary CredentialRequestOptions {
1394   FederatedCredentialRequestOptions federated;
1395 };
1396 </pre>
1397 <dl dfn-for="FederatedCredential">
1398   : <dfn attribute>provider</dfn>
1399   :: The credential's federated identity provider. See [[#provider-identification]] for
1400 details regarding valid formats.
1401
1402   : <dfn attribute>protocol</dfn>
1403   :: The credential's federated identity provider's protocol (e.g. "openidconnect"). If the
1404 value is null, then the protocol can be inferred from the
1405  FederatedCredential/provider.
1406
1407   :  Credential[[type]]
1408   :: The  FederatedCredential [=interface object=] has an internal slot named  [[type]] whose
1409 value is "dfn const federated".
1410
1411   :  Credential[[discovery]]
1412   :: The  FederatedCredential [=interface object=] has an internal slot named  [[discovery]]
1413 whose value is " Credential[[discovery]]/credential store".
1414
1415   : <dfn constructor>FederatedCredential(data)</dfn>
1416   :: This constructor accepts a  FederatedCredentialInit (ldata), and runs the following steps:
```

```
1408 request, and "Does Not Match" otherwise.
1409
1410 1. If options has a  CredentialRequestOptions/password member whose value is true, then
1411 return "Matches".
1412
1413 2. Return "Does Not Match".
1414
1415 </section>
1416
1417 <!--
1418
1419
1420
1421
1422
1423
1424
1425 -->
1426 <section>
1427 # Federated Credentials # {#federated}
1428
1429 ## The `FederatedCredential` Interface ## {#federatedcredential-interface}
1430
1431 <pre class="idl">
1432 [Constructor(FederatedCredentialInit data),
1433 Exposed=Window,
1434 SecureContext]
1435 interface FederatedCredential : Credential {
1436   readonly attribute USVString provider;
1437   readonly attribute DOMString? protocol;
1438 };
1439 FederatedCredential implements CredentialUserData;
1440
1441 dictionary FederatedCredentialRequestOptions {
1442   sequence<USVString> providers;
1443   sequence<DOMString> protocols;
1444 };
1445
1446 partial dictionary CredentialRequestOptions {
1447   FederatedCredentialRequestOptions federated;
1448 };
1449 </pre>
1450 <dl dfn-for="FederatedCredential">
1451   : <dfn attribute>provider</dfn>
1452   :: The credential's federated identity provider. See [[#provider-identification]] for
1453 details regarding valid formats.
1454
1455   : <dfn attribute>protocol</dfn>
1456   :: The credential's federated identity provider's protocol (e.g. "openidconnect"). If the
1457 value is null, then the protocol can be inferred from the
1458  FederatedCredential/provider.
1459
1460   :  Credential[[type]]
1461   :: The  FederatedCredential [=interface object=] has an internal slot named  [[type]] whose
1462 value is "dfn const federated".
1463
1464   :  Credential[[discovery]]
1465   :: The  FederatedCredential [=interface object=] has an internal slot named  [[discovery]]
1466 whose value is " Credential[[discovery]]/credential store".
1467
1468   : <dfn constructor>FederatedCredential(data)</dfn>
1469   :: This constructor accepts a  FederatedCredentialInit (ldata), and runs the following steps:
```

1417
1418 1. Let `lrl` be the result of executing `<a abstract-op>Create a `FederatedCredential` from`
1419 ``FederatedCredentialInit`` `` on `ldata`.
1420
1421 2. If `lrl` is an `[=exception=]`, `[=throw=]` `lrl`.
1422
1423 Otherwise, return `lrl`.
1424
1425 `</dl>`
1426 `{{FederatedCredential}}` objects can be created by passing a `{{FederatedCredentialInit}}` dictionary
1427 into `{{CredentialsContainer/create()|navigator.credentials.create()}}`.
1428
1429 `<pre class="idl">`
1430 `dictionary FederatedCredentialInit : CredentialData {`
1431 `USVString name;`
1432 `USVString iconURL;`
1433 `required USVString provider;`
1434 `DOMString protocol;`
1435 `};`
1436
1437 `partial dictionary CredentialCreationOptions {`
1438 `FederatedCredentialInit federated;`
1439 `};`
1440 `</pre>`
1441
1442 `{{FederatedCredential}}` objects are `[=Credential/origin bound=]`.
1443
1444 `{{FederatedCredential}}`'s `[=interface object=]` inherits `{{Credential}}`'s implementation of
1445 `{{Credential/[[DiscoverFromExternalSource]](origin, options)}}`, and defines its own implementation of
1446 `{{FederatedCredential/[[CollectFromCredentialStore]](origin, options)}}`,
1447 `{{FederatedCredential/[[Create]](options)}}`, and
1448 `{{FederatedCredential/[[Store]](credential)}}`.
1449
1450 Note: If, in the future, we teach the user agent to obtain authentication tokens on a user's
1451 behalf, we could do so by building an implementation of `[[DiscoverFromExternalSource]](origin,`
1452 `options)`.
1453
1454 `### Identifying Providers ### {#provider-identification}`
1455
1456 Every site should use the same identifier when referring to a specific federated identity
1457 provider. For example,
1458 `Facebook Login`
1459 shouldn't be referred to as "Facebook" and "Facebook Login" and "FB" and "FBL" and
1460 "Facebook.com"
1461 and so on. It should have a canonical identifier which everyone can make use of, as consistent
1462 identification makes it possible for user agents to be helpful.
1463
1464 For consistency, federations passed into the APIs defined in this document (e.g.
1465 `{{FederatedCredentialRequestOptions}}`'s `{{FederatedCredentialRequestOptions/providers}}` array, or
1466 `{{FederatedCredential}}`'s `{{FederatedCredential/provider}}` property) MUST be identified by the
1467 `ASCII serialization` of the origin the provider uses
1468 for sign in. That is, Facebook would be represented by `https://www.facebook.com` and Google by
1469 `https://accounts.google.com`.
1470
1471 This serialization of an `[=origin=]` does `_not_` include a trailing U+002F SOLIDUS ("`/`"), but
1472 user agents SHOULD accept them silently: `https://accounts.google.com/` is clearly
1473 intended to be the same as `https://accounts.google.com`.
1474
1475 `## Algorithms ## {#federatedcredential-algorithms}`
1476
1477 `<h4 algorithm id="collectfromcredentialstore-federatedcredential">`
1478 ``FederatedCredential`'s [[CollectFromCredentialStore]](origin, options)`
1479 `</h4>`
1480
1481 `<dfn for="FederatedCredential" method>[[CollectFromCredentialStore]](origin, options)</dfn>` is
1482 called
1483 with an `[=environment settings object/origin=]` (`loriginl`) and a `{{CredentialRequestOptions}}` (`l`
1484 `optionsl`),
1485 and returns a set of `{{Credential}}` objects from

1470
1471 1. Let `lrl` be the result of executing `<a abstract-op>Create a `FederatedCredential` from`
1472 ``FederatedCredentialInit`` `` on `ldata`.
1473
1474 2. If `lrl` is an `[=exception=]`, `[=throw=]` `lrl`.
1475
1476 Otherwise, return `lrl`.
1477
1478 `</dl>`
1479 `{{FederatedCredential}}` objects can be created by passing a `{{FederatedCredentialInit}}` dictionary
1480 into `{{CredentialsContainer/create()|navigator.credentials.create()}}`.
1481
1482 `<pre class="idl">`
1483 `dictionary FederatedCredentialInit : CredentialData {`
1484 `USVString name;`
1485 `USVString iconURL;`
1486 `required USVString provider;`
1487 `DOMString protocol;`
1488 `};`
1489
1490 `partial dictionary CredentialCreationOptions {`
1491 `FederatedCredentialInit federated;`
1492 `};`
1493 `</pre>`
1494
1495 `{{FederatedCredential}}` objects are `[=Credential/origin bound=]`.
1496
1497 `{{FederatedCredential}}`'s `[=interface object=]` inherits `{{Credential}}`'s implementation of
1498 `{{Credential/[[DiscoverFromExternalSource]](origin, options)}}`, and defines its own implementation of
1499 `{{FederatedCredential/[[CollectFromCredentialStore]](origin, options)}}`,
1500 `{{FederatedCredential/[[Create]](origin, options)}}`, and
1501 `{{FederatedCredential/[[Store]](credential)}}`.
1502
1503 Note: If, in the future, we teach the user agent to obtain authentication tokens on a user's
1504 behalf, we could do so by building an implementation of `[[DiscoverFromExternalSource]](origin,`
1505 `options)`.
1506
1507 `### Identifying Providers ### {#provider-identification}`
1508
1509 Every site should use the same identifier when referring to a specific federated identity
1510 provider. For example,
1511 `Facebook Login`
1512 shouldn't be referred to as "Facebook" and "Facebook Login" and "FB" and "FBL" and
1513 "Facebook.com"
1514 and so on. It should have a canonical identifier which everyone can make use of, as consistent
1515 identification makes it possible for user agents to be helpful.
1516
1517 For consistency, federations passed into the APIs defined in this document (e.g.
1518 `{{FederatedCredentialRequestOptions}}`'s `{{FederatedCredentialRequestOptions/providers}}` array, or
1519 `{{FederatedCredential}}`'s `{{FederatedCredential/provider}}` property) MUST be identified by the
1520 `ASCII serialization` of the origin the provider uses
1521 for sign in. That is, Facebook would be represented by `https://www.facebook.com` and Google by
1522 `https://accounts.google.com`.
1523
1524 This serialization of an `[=origin=]` does `_not_` include a trailing U+002F SOLIDUS ("`/`"), but
1525 user agents SHOULD accept them silently: `https://accounts.google.com/` is clearly
1526 intended to be the same as `https://accounts.google.com`.
1527
1528 `## Algorithms ## {#federatedcredential-algorithms}`
1529
1530 `<h4 algorithm id="collectfromcredentialstore-federatedcredential">`
1531 ``FederatedCredential`'s [[CollectFromCredentialStore]](origin, options)`
1532 `</h4>`
1533
1534 `<dfn for="FederatedCredential" method>[[CollectFromCredentialStore]](origin, options)</dfn>` is
1535 called
1536 with an `[=environment settings object/origin=]` (`loriginl`) and a `{{CredentialRequestOptions}}` (`l`
1537 `optionsl`),
1538 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: loptions["{{CredentialRequestOptions/federated}}"] [=map/exists=].
1487
1488   2. Return the result of <a abstract-op It="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 loriginl.
1493     3. If loptions["{{CredentialRequestOptions/federated}}"]
1494       ["{{FederatedCredentialRequestOptions/providers}}"]
1495       [=map/exists=], its value [=list/contains=] the credentials's {{FederatedCredential/provider}}.
1496     4. If loptions["{{CredentialRequestOptions/federated}}"]
1497       ["{{FederatedCredentialRequestOptions/protocols}}"]
1498       [=map/exists=], its value [=list/contains=] the credentials's {{FederatedCredential/protocol}}.
1499
1500 </ol>
1501
1502 <h4 algorithm id="create-federatedcredential">
1503   FederatedCredential's [[Create]](options)
1504 </h4>
1505
1506 <dfn for="FederatedCredential" method>[[Create]](options)</dfn> is called with a
1507 {{CredentialCreationOptions}} (loptionsl), and returns a {{FederatedCredential}} if one can be created,
1508 null otherwise, or an [=exception=] in exceptional circumstances:
1509
1510 <ol class="algorithm">
1511   1. Assert: loptions["{{CredentialCreationOptions/federated}}"] [=map/exists=].
1512
1513   2. Return the result of executing <a abstract-op>Create a `FederatedCredential` from
1514     FederatedCredentialInit </a> on loptions["{{CredentialCreationOptions/federated}}"].
1515
1516 </ol>
1517
1518 <h4 algorithm id="store-federatedcredential">
1519   FederatedCredential's [[Store]](credential)
1520 </h4>
1521
1522 <dfn for="FederatedCredential" method>[[Store]](credential)</dfn> is called with a
1523 {{FederatedCredential}} (lcredentiall), and returns once lcredentiall is persisted to the
1524 [=credential store=].
1525
1526 <ol class="algorithm">
1527   1. If the user agent's [=credential store=] contains a {{FederatedCredential}} whose
1528     {{Credential/id}} attribute is lcredentiall's {{Credential/id}} and whose {{[[origin]]}}
1529     slot is the [=same origin=] as lcredentiall's {{Credential/[origin]}}}, and
1530     whose {{FederatedCredential/provider}} is lcredentiall's
1531     {{FederatedCredential/provider}}, then return.
1532
1533   2. Store a {{FederatedCredential}} in the [=credential store=] with the following
1534     properties:
1535
1536     : {{Credential/id}}
1537     :: lcredentiall's {{Credential/id}}
1538     : {{CredentialUserData/name}}
1539     :: lcredentiall's {{CredentialUserData/name}}
1540     : {{CredentialUserData/iconURL}}
1541     :: lcredentiall's {{CredentialUserData/iconURL}}
1542     : {{Credential/[origin]}}
1543     :: lcredentiall's {{Credential/[origin]}}
1544     : {{FederatedCredential/provider}}
1545     :: lcredentiall's {{FederatedCredential/provider}}
1546     : {{FederatedCredential/protocol}}
1547     :: lcredentiall's {{FederatedCredential/protocol}}
1548
1549 </ol>
1550
1551 <h4 algorithm id="construct-federatedcredential-data">
1552   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: loptions["{{CredentialRequestOptions/federated}}"] [=map/exists=].
1540
1541   2. Return the result of <a abstract-op It="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 loriginl.
1546     3. If loptions["{{CredentialRequestOptions/federated}}"]
1547       ["{{FederatedCredentialRequestOptions/providers}}"]
1548       [=map/exists=], its value [=list/contains=] the credentials's {{FederatedCredential/provider}}.
1549     4. If loptions["{{CredentialRequestOptions/federated}}"]
1550       ["{{FederatedCredentialRequestOptions/protocols}}"]
1551       [=map/exists=], its value [=list/contains=] the credentials's {{FederatedCredential/protocol}}.
1552
1553 </ol>
1554
1555 <h4 algorithm id="create-federatedcredential">
1556   FederatedCredential's [[Create]](origin, options)
1557 </h4>
1558
1559 <dfn for="FederatedCredential" method>[[Create]](origin, options)</dfn> is called with
1560 an [=environment settings object/origin=] (<var ignore>origin</var>) and a
1561 {{CredentialCreationOptions}} (loptionsl), and returns a {{FederatedCredential}} if one can be created,
1562 null otherwise, or an [=exception=] in exceptional circumstances:
1563
1564 <ol class="algorithm">
1565   1. Assert: loptions["{{CredentialCreationOptions/federated}}"] [=map/exists=].
1566
1567   2. Return the result of executing <a abstract-op>Create a `FederatedCredential` from
1568     FederatedCredentialInit </a> on loptions["{{CredentialCreationOptions/federated}}"].
1569
1570 </ol>
1571
1572 <h4 algorithm id="store-federatedcredential">
1573   FederatedCredential's [[Store]](credential)
1574 </h4>
1575
1576 <dfn for="FederatedCredential" method>[[Store]](credential)</dfn> is called with a
1577 {{FederatedCredential}} (lcredentiall), and returns once lcredentiall is persisted to the
1578 [=credential store=].
1579
1580 <ol class="algorithm">
1581   1. If the user agent's [=credential store=] contains a {{FederatedCredential}} whose
1582     {{Credential/id}} attribute is lcredentiall's {{Credential/id}} and whose {{[[origin]]}}
1583     slot is the [=same origin=] as lcredentiall's {{Credential/[origin]}}}, and
1584     whose {{FederatedCredential/provider}} is lcredentiall's
1585     {{FederatedCredential/provider}}, then return.
1586
1587   2. Store a {{FederatedCredential}} in the [=credential store=] with the following
1588     properties:
1589
1590     : {{Credential/id}}
1591     :: lcredentiall's {{Credential/id}}
1592     : {{CredentialUserData/name}}
1593     :: lcredentiall's {{CredentialUserData/name}}
1594     : {{CredentialUserData/iconURL}}
1595     :: lcredentiall's {{CredentialUserData/iconURL}}
1596     : {{Credential/[origin]}}
1597     :: lcredentiall's {{Credential/[origin]}}
1598     : {{FederatedCredential/provider}}
1599     :: lcredentiall's {{FederatedCredential/provider}}
1600     : {{FederatedCredential/protocol}}
1601     :: lcredentiall's {{FederatedCredential/protocol}}
1602
1603 </ol>
1604
1605 <h4 algorithm id="construct-federatedcredential-data">
1606   Create a `FederatedCredential` from `FederatedCredentialInit
```

```

1548 </h4>
1549
1550 To <dfn abstract-op>Create a `FederatedCredential` from `FederatedCredentialInit` </dfn>, given a
1551 {{FederatedCredentialInit}} (init), run these steps.
1552
1553 <ol class="algorithm">
1554 1. Let lcl be a new {{FederatedCredential}} object.
1555
1556 2. If any of the following are the empty string, return a {{TypeError}} [=exception=]:
1557
1558   * init.{{CredentialData/id}}'s value
1559   * init.{{FederatedCredentialInit/provider}}'s value
1560
1561 3. Set lcl's properties as follows:
1562
1563   : {{Credential/id}}
1564   :: init.{{CredentialData/id}}'s value
1565   : {{FederatedCredential/provider}}
1566   :: init.{{FederatedCredentialInit/provider}}'s value
1567   : {{CredentialUserData/iconURL}}
1568   :: init.{{CredentialUserData/iconURL}}'s value
1569   : {{CredentialUserData/name}}
1570   :: init.{{CredentialUserData/name}}'s value
1571   : {{Credential/[[origin]]}}
1572   :: The [=environment settings object/origin=] of the [=current settings object=].
1573
1574 4. Return lcl.
1575 </ol>
1576
1577 </section>
1578
1579 <!--
1580
1581
1582
1583
1584
1585
1586
1587
1588 -->
1589 <section>
1590 # User Mediation # {#user-mediation}
1591
1592 Exposing credential information to the web via an API has a number of potential impacts on user
1593 privacy. The user agent, therefore, MUST involve the user in a number of cases in order to ensure
1594 that they clearly understands what's going on, and with whom their credentials are being shared.
1595
1596 We call a particular action <dfn export It="user mediateduser mediation">user mediated</dfn> if
1597 it takes place after gaining a user's explicit consent. Consent might be expressed through a
1598 user's direct interaction with a [=credential chooser=] interface, for example. In general, [=user
1599 mediated=] actions will involve presenting the user some sort of UI, and asking them to make a
1600 decision.
1601
1602 An action is unmediated if it takes place silently, without explicit user consent. For example,
1603 if a user configures their browser to grant persistent credential access to a particular origin,
1604 credentials may be provided without presenting the user with a UI requesting a decision.
1605
1606 Here we'll spell out a few requirements that hold for all credential types, but note that there's
1607 a good deal of latitude left up to the user agent (which is in a privileged position to assist
1608 the user). Moreover, specific credential types may have distinct requirements that exceed the
1609 requirements laid out more generally here.
1610
1611 ## Storing and Updating Credentials ## {#user-mediated-storage}
1612
1613 Credential information is sensitive data, and users MUST remain in control of that information's
1614 storage. Inadvertent credential storage could, for instance, unexpectedly link a user's local

```

```

1602 </h4>
1603
1604 To <dfn abstract-op>Create a `FederatedCredential` from `FederatedCredentialInit` </dfn>, given a
1605 {{FederatedCredentialInit}} (init), run these steps.
1606
1607 <ol class="algorithm">
1608 1. Let lcl be a new {{FederatedCredential}} object.
1609
1610 2. If any of the following are the empty string, return a {{TypeError}} [=exception=]:
1611
1612   * init.{{CredentialData/id}}'s value
1613   * init.{{FederatedCredentialInit/provider}}'s value
1614
1615 3. Set lcl's properties as follows:
1616
1617   : {{Credential/id}}
1618   :: init.{{CredentialData/id}}'s value
1619   : {{FederatedCredential/provider}}
1620   :: init.{{FederatedCredentialInit/provider}}'s value
1621   : {{CredentialUserData/iconURL}}
1622   :: init.{{CredentialUserData/iconURL}}'s value
1623   : {{CredentialUserData/name}}
1624   :: init.{{CredentialUserData/name}}'s value
1625   : {{Credential/[[origin]]}}
1626   :: The [=environment settings object/origin=] of the [=current settings object=].
1627
1628 4. Return lcl.
1629 </ol>
1630
1631 </section>
1632
1633 <!--
1634
1635
1636
1637
1638
1639
1640
1641
1642 -->
1643 <section>
1644 # User Mediation # {#user-mediation}
1645
1646 Exposing credential information to the web via an API has a number of potential impacts on user
1647 privacy. The user agent, therefore, MUST involve the user in a number of cases in order to ensure
1648 that they clearly understands what's going on, and with whom their credentials are being shared.
1649
1650 We call a particular action <dfn export It="user mediateduser mediation">user mediated</dfn> if
1651 it takes place after gaining a user's explicit consent. Consent might be expressed through a
1652 user's direct interaction with a [=credential chooser=] interface, for example. In general, [=user
1653 mediated=] actions will involve presenting the user some sort of UI, and asking them to make a
1654 decision.
1655
1656 An action is unmediated if it takes place silently, without explicit user consent. For example,
1657 if a user configures their browser to grant persistent credential access to a particular origin,
1658 credentials may be provided without presenting the user with a UI requesting a decision.
1659
1660 Here we'll spell out a few requirements that hold for all credential types, but note that there's
1661 a good deal of latitude left up to the user agent (which is in a privileged position to assist
1662 the user). Moreover, specific credential types may have distinct requirements that exceed the
1663 requirements laid out more generally here.
1664
1665 ## Storing and Updating Credentials ## {#user-mediated-storage}
1666
1667 Credential information is sensitive data, and users MUST remain in control of that information's
1668 storage. Inadvertent credential storage could, for instance, unexpectedly link a user's local

```

1615 profile on a particular device to a specific online persona. To mitigate the risk of surprise:
1616
1617 1. Credential information SHOULD NOT be stored or updated without [=user mediation=]. For
1618 example, the user agent could display a "Save this credential?" dialog box to the user in
1619 response to each call to {{store()}}.
1620
1621 User consent MAY be inferred if a user agent chooses to offer a persistent grant of consent
1622 in the form of an "Always save passwords" option (though we'd suggest that user agents should
1623 err on the side of something more narrowly scoped: perhaps "Always save _generated_
1624 passwords.", or "Always save passwords for this site.").

1625
1626 2. User agents SHOULD notify users when credentials are stored. This might take the form of an
1627 icon in the address bar, or some similar location.
1628
1629 3. User agents MUST allow users to manually remove stored credentials. This functionality might
1630 be implemented as a settings page, or via interaction with a notification as described above.
1631
1632 ## Requiring User Mediation ## {#user-mediation-requirement}
1633
1634 By default, [=user mediation=] is required for all [=origins=], as the relevant [=prevent silent
1635 access flag=] in the [=credential store=] is set to `true`. Users MAY choose to grant an
1636 origin persistent access to credentials (perhaps in the form of a "Stay signed into this site."
1637 option), which would set this flag to `false`. In this case, the user would always be signed into
1638 that site, which is desirable from the perspective of usability and convenience, but which
1639 might nevertheless have surprising implications (consider a user agent which syncs this flag's
1640 state across devices, for instance).

1641
1642 To mitigate the risk of surprise:
1643
1644 1. User agents MUST allow users to require [=user mediation=] for a given origin or for all
1645 origins. This functionality might be implemented as a global toggle that overrides each
1646 origin's <a>prevent silent access` flag to return `false`, or via more granular
1647 settings for specific origins (or specific credentials on specific origins).
1648
1649 2. User agents MUST NOT set an [=origin=]'s <a>prevent silent access` flag to
1650 `false` without [=user mediation=]. For example, the [=credential chooser=] described in
1651 [[#user-mediated-selection]] could have a checkbox which the user could toggle to mark a
1652 credential as available without mediation for the origin, or the user agent could have an
1653 onboarding process for its credential manager which asked a user for a default setting.
1654
1655 3. User agents MUST notify users when credentials are provided to an origin. This could take the
1656 form of an icon in the address bar, or some similar location.
1657
1658 4. If a user clears her browsing data for an origin (cookies, localStorage, and so on), the user
1659 agent MUST set the <a>prevent silent access` flag to `true` for that origin.
1660
1661 ## Credential Selection ## {#user-mediated-selection}
1662
1663 When responding to a call to {{CredentialsContainer/get()}} on an origin which requires
1664 [=user mediation=], user agents MUST ask the user for permission to share credential information.
1665 This SHOULD take the form of a <dfn export>credential chooser</dfn> which presents the user with a
1666 list of credentials that are available for use on a site, allowing them to select one which should
1667 be provided to the website, or to reject the request entirely.
1668
1669 The chooser interface SHOULD be implemented in such a way as to be distinguishable from UI which
1670 a website could produce. For example, the chooser might overlap the user agent's UI in some
1671 unspoofable way.
1672
1673 The chooser interface MUST include an indication of the origin which is requesting credentials.
1674
1675 The chooser interface SHOULD include all {{Credential}} objects associated with the origin that
1676 requested credentials.
1677
1678 User agents MAY internally associate information with each {{Credential}} object beyond the
1679 attributes specified in this document in order to enhance the utility of such a chooser. For
1680 example, favicons could help disambiguate identity providers, etc. Any additional information
1681 stored MUST not be exposed directly to the web.
1682

1669 profile on a particular device to a specific online persona. To mitigate the risk of surprise:
1670
1671 1. Credential information SHOULD NOT be stored or updated without [=user mediation=]. For
1672 example, the user agent could display a "Save this credential?" dialog box to the user in
1673 response to each call to {{store()}}.
1674
1675 User consent MAY be inferred if a user agent chooses to offer a persistent grant of consent
1676 in the form of an "Always save passwords" option (though we'd suggest that user agents should
1677 err on the side of something more narrowly scoped: perhaps "Always save _generated_
1678 passwords.", or "Always save passwords for this site.").

1679
1680 2. User agents SHOULD notify users when credentials are stored. This might take the form of an
1681 icon in the address bar, or some similar location.
1682
1683 3. User agents MUST allow users to manually remove stored credentials. This functionality might
1684 be implemented as a settings page, or via interaction with a notification as described above.
1685
1686 ## Requiring User Mediation ## {#user-mediation-requirement}
1687
1688 By default, [=user mediation=] is required for all [=origins=], as the relevant [=prevent silent
1689 access flag=] in the [=credential store=] is set to `true`. Users MAY choose to grant an
1690 origin persistent access to credentials (perhaps in the form of a "Stay signed into this site."
1691 option), which would set this flag to `false`. In this case, the user would always be signed into
1692 that site, which is desirable from the perspective of usability and convenience, but which
1693 might nevertheless have surprising implications (consider a user agent which syncs this flag's
1694 state across devices, for instance).

1695
1696 To mitigate the risk of surprise:
1697
1698 1. User agents MUST allow users to require [=user mediation=] for a given origin or for all
1699 origins. This functionality might be implemented as a global toggle that overrides each
1700 origin's <a>prevent silent access` flag to return `false`, or via more granular
1701 settings for specific origins (or specific credentials on specific origins).
1702
1703 2. User agents MUST NOT set an [=origin=]'s <a>prevent silent access` flag to
1704 `false` without [=user mediation=]. For example, the [=credential chooser=] described in
1705 [[#user-mediated-selection]] could have a checkbox which the user could toggle to mark a
1706 credential as available without mediation for the origin, or the user agent could have an
1707 onboarding process for its credential manager which asked a user for a default setting.
1708
1709 3. User agents MUST notify users when credentials are provided to an origin. This could take the
1710 form of an icon in the address bar, or some similar location.
1711
1712 4. If a user clears her browsing data for an origin (cookies, localStorage, and so on), the user
1713 agent MUST set the <a>prevent silent access` flag to `true` for that origin.
1714
1715 ## Credential Selection ## {#user-mediated-selection}
1716
1717 When responding to a call to {{CredentialsContainer/get()}} on an origin which requires
1718 [=user mediation=], user agents MUST ask the user for permission to share credential information.
1719 This SHOULD take the form of a <dfn export>credential chooser</dfn> which presents the user with a
1720 list of credentials that are available for use on a site, allowing them to select one which should
1721 be provided to the website, or to reject the request entirely.
1722
1723 The chooser interface SHOULD be implemented in such a way as to be distinguishable from UI which
1724 a website could produce. For example, the chooser might overlap the user agent's UI in some
1725 unspoofable way.
1726
1727 The chooser interface MUST include an indication of the origin which is requesting credentials.
1728
1729 The chooser interface SHOULD include all {{Credential}} objects associated with the origin that
1730 requested credentials.
1731
1732 User agents MAY internally associate information with each {{Credential}} object beyond the
1733 attributes specified in this document in order to enhance the utility of such a chooser. For
1734 example, favicons could help disambiguate identity providers, etc. Any additional information
1735 stored MUST not be exposed directly to the web.
1736

```

1683 The chooser's behavior is not defined here: user agents are encouraged to experiment with UI
1684 treatments that educate users about their authentication options, and guide them through the
1685 process of choosing a credential to present. That said, the interface to the chooser is as
1686 follows:
1687
1688 <section algorithm="ask the user to choose">
1689   The user agent can <dfn export abstract-op local-It="ask to choose">ask the user to choose a
1690   `Credential` </dfn>, given a {{CredentialRequestOptions}} (loptionsl), and a set of
1691   {{Credential}} objects from the [=credential store=] (llocally discovered credentialsl).
1692
1693   This algorithm returns either `null` if the user chose not to share a credential with the site,
1694   a {{Credential}} object if the user chose a specific credential, or a {{Credential}} [=interface
1695   object=] if the user chose a type of credential.
1696
1697   <div class="note">
1698     It seems reasonable for the chooser interface to display the list of llocally discovered
1699     credentialsl to the user, perhaps something like this exceptionally non-normative mock:
1700
1701     
1702
1703     If the loptionsl provided is not <a>matchable <i lang="la">a priori</i></a>, then it might
1704     also make sense for the chooser interface to list the [=relevant credential interface
1705     objects=] for loptionsl that aren't covered by the list of explicit credentials. If, for
1706     instance, a site accepts webauthn-style authenticators, then "Security Key" might show up
1707     in the chooser list with an appropriate icon.
1708
1709     Also, note that in some cases the user agent may skip the chooser entirely. For example, if
1710     the only [=relevant credential interface objects=] is one that itself requires user
1711     interaction, the user agent may return that interface directly, and rely on its internal
1712     mediation flow for user consent.
1713   </div>
1714 </section>
1715 </section>
1716
1717 <!--
1718
1719
1720
1721
1722
1723
1724
1725 -->
1726 <section>
1727   # Security and Privacy Considerations # {#security-and-privacy}
1728
1729   The following sections represent guidelines for various security and privacy considerations.
1730   Individual credential types may enforce stricter or more relaxed versions of these guidelines.
1731
1732   ## Cross-domain credential access ## {#security-credential-access}
1733
1734   Credentials are sensitive information, and user agents need to exercise caution in determining
1735   when they can be safely shared with a website. The safest option is to restrict credential
1736   sharing to the exact origin on which they were saved. That is likely too restrictive for the
1737   web, however: consider sites which divide functionality into subdomains like `example.com` vs
1738   `admin.example.com`.
1739
1740   As a compromise between annoying users, and securing their credentials, user agents:
1741
1742   1. MUST NOT share credentials between origins whose scheme components represent a downgrade
1743   in
1744   security. That is, it may make sense to allow credentials saved on `http://example.com/` to
1745   be made available to `https://example.com/` (in order to encourage developers to migrate to
1746   secure transport), but the inverse would be dangerous.
1747
1748   2. MAY use the Public Suffix List [[PSL]] to determine the effective scope of a credential by
1749   comparing the [=registerable domain=] of the credential's {{Credential/[origin]}} with the
1750   origin in which {{CredentialsContainer/get()}} is called. That is: credentials saved on

```

```

1737 The chooser's behavior is not defined here: user agents are encouraged to experiment with UI
1738 treatments that educate users about their authentication options, and guide them through the
1739 process of choosing a credential to present. That said, the interface to the chooser is as
1740 follows:
1741
1742 <section algorithm="ask the user to choose">
1743   The user agent can <dfn export abstract-op local-It="ask to choose">ask the user to choose a
1744   `Credential` </dfn>, given a {{CredentialRequestOptions}} (loptionsl), and a set of
1745   {{Credential}} objects from the [=credential store=] (llocally discovered credentialsl).
1746
1747   This algorithm returns either `null` if the user chose not to share a credential with the site,
1748   a {{Credential}} object if the user chose a specific credential, or a {{Credential}} [=interface
1749   object=] if the user chose a type of credential.
1750
1751   <div class="note">
1752     It seems reasonable for the chooser interface to display the list of llocally discovered
1753     credentialsl to the user, perhaps something like this exceptionally non-normative mock:
1754
1755     
1756
1757     If the loptionsl provided is not <a>matchable <i lang="la">a priori</i></a>, then it might
1758     also make sense for the chooser interface to list the [=relevant credential interface
1759     objects=] for loptionsl that aren't covered by the list of explicit credentials. If, for
1760     instance, a site accepts webauthn-style authenticators, then "Security Key" might show up
1761     in the chooser list with an appropriate icon.
1762
1763     Also, note that in some cases the user agent may skip the chooser entirely. For example, if
1764     the only [=relevant credential interface objects=] is one that itself requires user
1765     interaction, the user agent may return that interface directly, and rely on its internal
1766     mediation flow for user consent.
1767   </div>
1768 </section>
1769 </section>
1770
1771 <!--
1772
1773
1774
1775
1776
1777
1778
1779 -->
1780 <section>
1781   # Security and Privacy Considerations # {#security-and-privacy}
1782
1783   The following sections represent guidelines for various security and privacy considerations.
1784   Individual credential types may enforce stricter or more relaxed versions of these guidelines.
1785
1786   ## Cross-domain credential access ## {#security-credential-access}
1787
1788   Credentials are sensitive information, and user agents need to exercise caution in determining
1789   when they can be safely shared with a website. The safest option is to restrict credential
1790   sharing to the exact origin on which they were saved. That is likely too restrictive for the
1791   web, however: consider sites which divide functionality into subdomains like `example.com` vs
1792   `admin.example.com`.
1793
1794   As a compromise between annoying users, and securing their credentials, user agents:
1795
1796   1. MUST NOT share credentials between origins whose scheme components represent a downgrade
1797   in
1798   security. That is, it may make sense to allow credentials saved on `http://example.com/` to
1799   be made available to `https://example.com/` (in order to encourage developers to migrate to
1800   secure transport), but the inverse would be dangerous.
1801
1802   2. MAY use the Public Suffix List [[PSL]] to determine the effective scope of a credential by
1803   comparing the [=registerable domain=] of the credential's {{Credential/[origin]}} with the
1804   origin in which {{CredentialsContainer/get()}} is called. That is: credentials saved on

```

```

1750 `https://admin.example.com/` and `https://example.com/` MAY be offered to users when
1751 {{CredentialsContainer/get()}} is called from `https://www.example.com/`, and vice versa.
1752
1753 3. MUST NOT offer credentials to an origin in response to {{CredentialsContainer/get()}} without
1754 [=user mediation=] if the credential's origin is not an exact match for the calling origin.
1755 That is, {{Credential}} objects for `https://example.com` would not be returned directly to
1756 `https://www.example.com`, but could be offered to the user via the chooser.
1757
1758 ## Credential Leakage ## {#security-leakage}
1759
1760 Developers are well-advised to take
1761 some precautions to mitigate the risk that a cross-site scripting attack could turn into
1762 persistent access to a user's account by setting a reasonable Content Security Policy [[!CSP]]
1763 which restricts the endpoints to which data can be sent. In particular, developers should ensure
1764 that the following directives are set, explicitly or implicitly, in their pages' policies:
1765
1766 * <a>script-src</a> and <a>object-src</a> both restrict script execution on a page, making
1767 it less likely that a cross-site scripting attack will succeed in the first place. If sites
1768 are populating <form> elements, also <a>form-action</a> directives should be set.
1769
1770 * <a>connect-src</a> restricts the origins to which {{fetch()}} may submit data (which
1771 mitigates the risk that credentials could be exfiltrated to `evil.com` .
1772
1773 * <a>child-src</a> restricts the nested browsing contexts which may be embedded in a page,
1774 making it more difficult to inject a malicious postMessage() target. [[WEBMESSAGING]]
1775
1776 Developers should, of course, also properly escape input and output, and consider using other
1777 layers of defense, such as Subresource Integrity [[SRI]] to further reduce risk.
1778
1779 When defining specific credential types, specific credential types SHOULD give due consideration
1780 to the ways in which credential data can be transmitted over the wire. It might be reasonable,
1781 for example, to define transmission mechanisms which are restricted to same-origin endpoints.
1782
1783 ## Insecure Sites ## {#insecure-sites}
1784
1785 User agents MUST NOT expose the APIs defined here to environments which are not [=secure
1786 contexts=]. User agents might implement autofill mechanisms which store user credentials and fill
1787 sign-in forms on non-<a><i>a priori</i> authenticated URLs</a>, but those sites cannot
1788 be trusted to interact directly with the credential manager in any meaningful way, and those sites
1789 MUST NOT have access to credentials saved in [=secure contexts=].
1790
1791 ## Origin Confusion ## {#security-origin-confusion}
1792
1793 If framed pages have access to the APIs defined here, it might be possible to confuse a user into
1794 granting access to credentials for an origin other than the <a>top-level browsing context</a>,
1795 which is the only security origin which users can reasonably be expected to understand.
1796
1797 Therefore, both {{get()}} and {{store()}} will result in a rejected {{Promise}} when called from
1798 a [=nested browsing context=], and the API and related interfaces are not exposed inside
1799 {{Worker}} contexts.
1800
1801 The algorithms in [[#algorithm-request]] and [[#algorithm-store]] contain more detail.
1802
1803 ## Timing Attacks ## {#security-timing}
1804
1805 If the user has no credentials for an origin, a call to {{CredentialsContainer/get()}} will
1806 resolve very quickly indeed. A malicious website could distinguish between a user with no
1807 credentials and a user with credentials who chooses not to share them.
1808
1809 User agents SHOULD also rate-limit credential requests. It's almost certainly abusive for a page
1810 to request credentials more than a few times in a short period.
1811
1812 ## Signing-Out ## {#security-signout}
1813
1814 If a user has chosen to automatically sign-in to websites, as discussed in
1815 [[#user-mediation-requirement]], then the user agent will provide credentials to an origin
1816 whenever it asks for them. The website can instruct the user agent to suppress this behavior by
1817 calling {{CredentialsContainer}}'s {{CredentialsContainer/preventSilentAccess()}} method, which
1818 will turn off automatic sign-in for a given origin.

```

```

1804 `https://admin.example.com/` and `https://example.com/` MAY be offered to users when
1805 {{CredentialsContainer/get()}} is called from `https://www.example.com/`, and vice versa.
1806
1807 3. MUST NOT offer credentials to an origin in response to {{CredentialsContainer/get()}} without
1808 [=user mediation=] if the credential's origin is not an exact match for the calling origin.
1809 That is, {{Credential}} objects for `https://example.com` would not be returned directly to
1810 `https://www.example.com`, but could be offered to the user via the chooser.
1811
1812 ## Credential Leakage ## {#security-leakage}
1813
1814 Developers are well-advised to take
1815 some precautions to mitigate the risk that a cross-site scripting attack could turn into
1816 persistent access to a user's account by setting a reasonable Content Security Policy [[!CSP]]
1817 which restricts the endpoints to which data can be sent. In particular, developers should ensure
1818 that the following directives are set, explicitly or implicitly, in their pages' policies:
1819
1820 * <a>script-src</a> and <a>object-src</a> both restrict script execution on a page, making
1821 it less likely that a cross-site scripting attack will succeed in the first place. If sites
1822 are populating <form> elements, also <a>form-action</a> directives should be set.
1823
1824 * <a>connect-src</a> restricts the origins to which {{fetch()}} may submit data (which
1825 mitigates the risk that credentials could be exfiltrated to `evil.com` .
1826
1827 * <a>child-src</a> restricts the nested browsing contexts which may be embedded in a page,
1828 making it more difficult to inject a malicious postMessage() target. [[WEBMESSAGING]]
1829
1830 Developers should, of course, also properly escape input and output, and consider using other
1831 layers of defense, such as Subresource Integrity [[SRI]] to further reduce risk.
1832
1833 When defining specific credential types, specific credential types SHOULD give due consideration
1834 to the ways in which credential data can be transmitted over the wire. It might be reasonable,
1835 for example, to define transmission mechanisms which are restricted to same-origin endpoints.
1836
1837 ## Insecure Sites ## {#insecure-sites}
1838
1839 User agents MUST NOT expose the APIs defined here to environments which are not [=secure
1840 contexts=]. User agents might implement autofill mechanisms which store user credentials and fill
1841 sign-in forms on non-<a><i>a priori</i> authenticated URLs</a>, but those sites cannot
1842 be trusted to interact directly with the credential manager in any meaningful way, and those sites
1843 MUST NOT have access to credentials saved in [=secure contexts=].
1844
1845 ## Origin Confusion ## {#security-origin-confusion}
1846
1847 If framed pages have access to the APIs defined here, it might be possible to confuse a user into
1848 granting access to credentials for an origin other than the <a>top-level browsing context</a>,
1849 which is the only security origin which users can reasonably be expected to understand.
1850
1851 Therefore, both {{get()}} and {{store()}} will result in a rejected {{Promise}} when called from
1852 a [=nested browsing context=], and the API and related interfaces are not exposed inside
1853 {{Worker}} contexts.
1854
1855 The algorithms in [[#algorithm-request]] and [[#algorithm-store]] contain more detail.
1856
1857 ## Timing Attacks ## {#security-timing}
1858
1859 If the user has no credentials for an origin, a call to {{CredentialsContainer/get()}} will
1860 resolve very quickly indeed. A malicious website could distinguish between a user with no
1861 credentials and a user with credentials who chooses not to share them.
1862
1863 User agents SHOULD also rate-limit credential requests. It's almost certainly abusive for a page
1864 to request credentials more than a few times in a short period.
1865
1866 ## Signing-Out ## {#security-signout}
1867
1868 If a user has chosen to automatically sign-in to websites, as discussed in
1869 [[#user-mediation-requirement]], then the user agent will provide credentials to an origin
1870 whenever it asks for them. The website can instruct the user agent to suppress this behavior by
1871 calling {{CredentialsContainer}}'s {{CredentialsContainer/preventSilentAccess()}} method, which
1872 will turn off automatic sign-in for a given origin.

```

```
1819 The user agent relies on the website to do the right thing; an inattentive (or malicious) website
1820 could simply neglect to call this method, causing the user agent to continue providing credentials
1821 against the user's apparent intention. This is marginally worse than the status-quo of a site that
1822 doesn't clear user credentials when they click "Sign-out", as the user agent becomes complicit in
1823 the authentication.
1824
1825 The user MUST have some control over this behavior. As noted in [[#user-mediation-requirement]],
1826 clearing cookies for an origin will also reset that origin's <a> prevent silent access flag</a>
1827 the [=credential store=] to `true`. Additionally, the user agent SHOULD provide some UI affordance
1828 for disabling automatic sign-in for a particular origin. This could be tied to the notification
1829 that credentials have been provided to an origin, for example.
1830
1831 ## Chooser Leakage ## {#security-chooser-leakage}
1832
1833 If a user agent's [=credential chooser=] displays images supplied by an origin (for example, if a
1834 {{Credential}} displays a site's favicon), then, requests for these images MUST NOT be directly
1835 tied to instantiating the chooser in order to avoid leaking chooser usage. One option would be to
1836 fetch the images in the background when saving or updating a {{Credential}}, and to cache them for
1837 the lifetime of the {{Credential}}.
1838
1839 These images MUST be fetched with the [=request/credentials mode=] set to `omit`, the
1840 [=request/service-workers mode=] set to `none`, the [=request/client=] set to `null`, the
1841 [=request/initiator=] set to the empty string, and the [=request/destination=] `subresource`.
1842
1843 Moreover, if the user agent allows the user to change either the name or icon associated with the
1844 credential, the alterations to the data SHOULD NOT be exposed to the website (consider a user who
1845 names two credentials for an origin "My fake account" and "My real account", for instance).
1846
1847 ## Locally Stored Data ## {#security-local-data}
1848
1849 This API offers an [=origin=] the ability to store data persistently along with a user's profile.
1850 Since most user agents treat credential data differently than "browsing data" (cookies, etc.)
1851 this might have the side effect of surprising a user who might believe that all traces of an
1852 origin have been wiped out when they clear their cookies.
1853
1854 User agents SHOULD provide UI that makes it clear to a user that credential data is stored for an
1855 origin, and SHOULD make it easy for users to remove such data when they're no longer interested in
1856 keeping it around.
1857 </section>
1858 <!--
1859 [Redacted content]
1860 -->
1861
1862 <section>
1863 # Implementation Considerations # {#implementation}
1864
1865 <em>This section is non-normative.</em>
1866
1867 ## Website Authors ## {#implementation-authors}
1868
1869 ISSUE(w3c/webappsec#290): Add some thoughts here about when and how the API
1870 should be used, especially with regard to {{CredentialRequestOptions/mediation}}.
1871
1872 ISSUE: Describe encoding restrictions of submitting credentials by {{fetch()}} with
1873 a {{FormData}} body.
1874
```

```
1873 The user agent relies on the website to do the right thing; an inattentive (or malicious) website
1874 could simply neglect to call this method, causing the user agent to continue providing credentials
1875 against the user's apparent intention. This is marginally worse than the status-quo of a site that
1876 doesn't clear user credentials when they click "Sign-out", as the user agent becomes complicit in
1877 the authentication.
1878
1879 The user MUST have some control over this behavior. As noted in [[#user-mediation-requirement]],
1880 clearing cookies for an origin will also reset that origin's <a> prevent silent access flag</a>
1881 the [=credential store=] to `true`. Additionally, the user agent SHOULD provide some UI affordance
1882 for disabling automatic sign-in for a particular origin. This could be tied to the notification
1883 that credentials have been provided to an origin, for example.
1884
1885 ## Chooser Leakage ## {#security-chooser-leakage}
1886
1887 If a user agent's [=credential chooser=] displays images supplied by an origin (for example, if a
1888 {{Credential}} displays a site's favicon), then, requests for these images MUST NOT be directly
1889 tied to instantiating the chooser in order to avoid leaking chooser usage. One option would be to
1890 fetch the images in the background when saving or updating a {{Credential}}, and to cache them for
1891 the lifetime of the {{Credential}}.
1892
1893 These images MUST be fetched with the [=request/credentials mode=] set to `omit`, the
1894 [=request/service-workers mode=] set to `none`, the [=request/client=] set to `null`, the
1895 [=request/initiator=] set to the empty string, and the [=request/destination=] `subresource`.
1896
1897 Moreover, if the user agent allows the user to change either the name or icon associated with the
1898 credential, the alterations to the data SHOULD NOT be exposed to the website (consider a user who
1899 names two credentials for an origin "My fake account" and "My real account", for instance).
1900
1901 ## Locally Stored Data ## {#security-local-data}
1902
1903 This API offers an [=origin=] the ability to store data persistently along with a user's profile.
1904 Since most user agents treat credential data differently than "browsing data" (cookies, etc.)
1905 this might have the side effect of surprising a user who might believe that all traces of an
1906 origin have been wiped out when they clear their cookies.
1907
1908 User agents SHOULD provide UI that makes it clear to a user that credential data is stored for an
1909 origin, and SHOULD make it easy for users to remove such data when they're no longer interested in
1910 keeping it around.
1911 </section>
1912 <!--
1913 [Redacted content]
1914 -->
1915
1916 <section>
1917 # Implementation Considerations # {#implementation}
1918
1919 <em>This section is non-normative.</em>
1920
1921 ## Website Authors ## {#implementation-authors}
1922
1923 ISSUE(w3c/webappsec#290): Add some thoughts here about when and how the API
1924 should be used, especially with regard to {{CredentialRequestOptions/mediation}}.
1925
1926 ISSUE: Describe encoding restrictions of submitting credentials by {{fetch()}} with
1927 a {{FormData}} body.
1928
```

```
1881 ## Extension Points ## {#implementation-extension}
1882
1883
1884 This document provides a generic, high-level API that's meant to be extended with specific types
1885 of [=credentials=] that serve specific authentication needs. Doing so is, hopefully,
1886 straightforward:
1887
1888 1. Define a new interface that inherits from {{Credential}}:
1889
1890 <div class="example">
1891 <pre>
1892   [Exposed=Window,
1893    SecureContext]
1894   interface ExampleCredential : Credential {
1895     // Definition goes here.
1896   };
1897 </pre>
1898 </div>
1899
1900 2. Define appropriate {{Credential/[[Create]](options)}},
1901 {{Credential/[[CollectFromCredentialStore]](origin, options)}},
1902 {{Credential/[[DiscoverFromExternalSource]](origin, options)}}, and
1903 {{Credential/[[Store]](credential)}} methods on `ExampleCredential`'s
1904 [=interface object=]. {{Credential/[[CollectFromCredentialStore]](origin, options)}}
1905 is appropriate for [=credentials=] that remain [=effective=] forever and
1906 can therefore simply be copied out of the [=credential store=], while
1907 {{Credential/[[DiscoverFromExternalSource]](origin, options)}} is appropriate for
1908 [=credentials=] that need to be re-generated from a [=credential source=].
1909
1910 <div class="example">
1911   `ExampleCredential`'s `[[CollectFromCredentialStore]](origin, options)` internal method is called
1912   with an [=environment settings object/origin=] ( `origin` ) and a `CredentialRequestOptions`
1913   object ( `options` ), and returns a set of {{Credential}}
1914   objects that match the options provided. If no matching {{Credential}} objects are
1915   available, the returned set will be empty.
1916
1917   1. Assert: `options` [ `example` ] exists.
1918
1919   2. If `options` [ `example` ] is not truthy, return the empty set.
1920
1921   3. For each <i>credential</i> in the [=credential store=]:
1922     1. ...
1923 </div>
1924
1925 3. Define the value of the `ExampleCredential` [=interface object=]'s {{{[type]}}} slot:
1926
1927 <div class="example">
1928   The `ExampleCredential` [=interface object=] has an internal slot named `[[type]]` whose
1929   value is the string "example".
1930 </div>
1931
1932 4. Define the value of the `ExampleCredential` [=interface object=]'s {{{[discovery]}}} slot:
1933
1934 <div class="example">
1935   The `ExampleCredential` [=interface object=] has an internal slot named `[[type]]` whose
1936   value is "{{Credential/[[discovery]]/credential store}}".
1937 </div>
1938
1939 4. Extend {{CredentialRequestOptions}} with the options the new credential type needs to respond
1940 reasonably to {{get()}}:
1941
1942 <div class="example">
1943 <pre>
1944   dictionary ExampleCredentialRequestOptions {
1945     // Definition goes here.
1946   };
1947
1948   partial dictionary CredentialRequestOptions {
```

```
1935 ## Extension Points ## {#implementation-extension}
1936
1937
1938 This document provides a generic, high-level API that's meant to be extended with specific types
1939 of [=credentials=] that serve specific authentication needs. Doing so is, hopefully,
1940 straightforward:
1941
1942 1. Define a new interface that inherits from {{Credential}}:
1943
1944 <div class="example">
1945 <pre>
1946   [Exposed=Window,
1947    SecureContext]
1948   interface ExampleCredential : Credential {
1949     // Definition goes here.
1950   };
1951 </pre>
1952 </div>
1953
1954 2. Define appropriate {{Credential/[[Create]](origin, options)}},
1955 {{Credential/[[CollectFromCredentialStore]](origin, options)}},
1956 {{Credential/[[DiscoverFromExternalSource]](origin, options)}}, and
1957 {{Credential/[[Store]](credential)}} methods on `ExampleCredential`'s
1958 [=interface object=]. {{Credential/[[CollectFromCredentialStore]](origin, options)}}
1959 is appropriate for [=credentials=] that remain [=effective=] forever and
1960 can therefore simply be copied out of the [=credential store=], while
1961 {{Credential/[[DiscoverFromExternalSource]](origin, options)}} is appropriate for
1962 [=credentials=] that need to be re-generated from a [=credential source=].
1963
1964 <div class="example">
1965   `ExampleCredential`'s `[[CollectFromCredentialStore]](origin, options)` internal method is called
1966   with an [=environment settings object/origin=] ( `origin` ) and a `CredentialRequestOptions`
1967   object ( `options` ), and returns a set of {{Credential}}
1968   objects that match the options provided. If no matching {{Credential}} objects are
1969   available, the returned set will be empty.
1970
1971   1. Assert: `options` [ `example` ] exists.
1972
1973   2. If `options` [ `example` ] is not truthy, return the empty set.
1974
1975   3. For each <i>credential</i> in the [=credential store=]:
1976     1. ...
1977 </div>
1978
1979 3. Define the value of the `ExampleCredential` [=interface object=]'s {{{[type]}}} slot:
1980
1981 <div class="example">
1982   The `ExampleCredential` [=interface object=] has an internal slot named `[[type]]` whose
1983   value is the string "example".
1984 </div>
1985
1986 4. Define the value of the `ExampleCredential` [=interface object=]'s {{{[discovery]}}} slot:
1987
1988 <div class="example">
1989   The `ExampleCredential` [=interface object=] has an internal slot named `[[type]]` whose
1990   value is "{{Credential/[[discovery]]/credential store}}".
1991 </div>
1992
1993 4. Extend {{CredentialRequestOptions}} with the options the new credential type needs to respond
1994 reasonably to {{get()}}:
1995
1996 <div class="example">
1997 <pre>
1998   dictionary ExampleCredentialRequestOptions {
1999     // Definition goes here.
2000   };
2001
2002   partial dictionary CredentialRequestOptions {
```

```

1950 ExampleCredentialRequestOptions example;
1951 };
1952 </pre>
1953 </div>
1954
1955 5. Extend {{CredentialCreationOptions}} with the data the new credential type needs to create
1956 `Credential` objects in response to {{create()}}:
1957
1958 <div class="example">
1959 <pre>
1960 dictionary ExampleCredentialInit {
1961 // Definition goes here.
1962 };
1963
1964 partial dictionary CredentialCreationOptions {
1965 ExampleCredentialInit example;
1966 };
1967 </pre>
1968 </div>
1969
1970 You might also find that new primitives are necessary. For instance, you might want to return
1971 many {{Credential}} objects rather than just one in some sort of complicated, multi-factor
1972 sign-in process. That might be accomplished in a generic fashion by adding a `getAll()` method to
1973 {{CredentialsContainer}} which returned a `sequence<Credential>`, and defining a reasonable
1974 mechanism for dealing with requesting credentials of distinct types.
1975
1976 For any such extension, we recommend getting in touch with
1977 <a href="mailto:public-webappsec@w3.org">public-webappsec@</a> for consultation and review.
1978
1979 ## Browser Extensions ## {#browser-extensions}
1980
1981 Ideally, user agents that implement an extension system of some sort will
1982 allow third-parties to hook into these API endpoints in order to improve
1983 the behavior of third party credential management software in the same way
1984 that user agents can improve their own via this imperative approach.
1985
1986 This could range from a complex new API that the user agent mediates, or
1987 simply by allowing extensions to overwrite the {{CredentialsContainer/get()}}
1988 and {{CredentialsContainer/store()}} endpoints for their own purposes.
1989 </section>
1990
1991 <!--
1992
1993
1994
1995
1996
1997
1998
1999 -->
2000 <section>
2001 # Future Work # {#teh-futur}
2002
2003 <em>This section is non-normative.</em>
2004
2005 The API defined here does the bare minimum to expose user agent's credential managers to the web,
2006 and allows the web to help those credential managers understand when federated identity providers
2007 are in use. The next logical step will be along the lines sketched in documents like [[WEB-LOGIN]]
2008 (and, to some extent, Mozilla's BrowserID [[BROWSERID]]).
2009
2010 The user agent is in the unique position of being able to effectively mediate the relationship
2011 between users, identity providers, and websites. If the user agent can remove some of the risk and
2012 confusion associated with the typical authentication flows, users will be in a significantly
2013 better position than today.
2014
2015 A natural way to expose this information might be to extend the {{FederatedCredential}} interface

```

```

2004 ExampleCredentialRequestOptions example;
2005 };
2006 </pre>
2007 </div>
2008
2009 5. Extend {{CredentialCreationOptions}} with the data the new credential type needs to create
2010 `Credential` objects in response to {{create()}}:
2011
2012 <div class="example">
2013 <pre>
2014 dictionary ExampleCredentialInit {
2015 // Definition goes here.
2016 };
2017
2018 partial dictionary CredentialCreationOptions {
2019 ExampleCredentialInit example;
2020 };
2021 </pre>
2022 </div>
2023
2024 You might also find that new primitives are necessary. For instance, you might want to return
2025 many {{Credential}} objects rather than just one in some sort of complicated, multi-factor
2026 sign-in process. That might be accomplished in a generic fashion by adding a `getAll()` method to
2027 {{CredentialsContainer}} which returned a `sequence<Credential>`, and defining a reasonable
2028 mechanism for dealing with requesting credentials of distinct types.
2029
2030 For any such extension, we recommend getting in touch with
2031 <a href="mailto:public-webappsec@w3.org">public-webappsec@</a> for consultation and review.
2032
2033 ## Browser Extensions ## {#browser-extensions}
2034
2035 Ideally, user agents that implement an extension system of some sort will
2036 allow third-parties to hook into these API endpoints in order to improve
2037 the behavior of third party credential management software in the same way
2038 that user agents can improve their own via this imperative approach.
2039
2040 This could range from a complex new API that the user agent mediates, or
2041 simply by allowing extensions to overwrite the {{CredentialsContainer/get()}}
2042 and {{CredentialsContainer/store()}} endpoints for their own purposes.
2043 </section>
2044
2045 <!--
2046
2047
2048
2049
2050
2051
2052
2053 -->
2054 <section>
2055 # Future Work # {#teh-futur}
2056
2057 <em>This section is non-normative.</em>
2058
2059 The API defined here does the bare minimum to expose user agent's credential managers to the web,
2060 and allows the web to help those credential managers understand when federated identity providers
2061 are in use. The next logical step will be along the lines sketched in documents like [[WEB-LOGIN]]
2062 (and, to some extent, Mozilla's BrowserID [[BROWSERID]]).
2063
2064 The user agent is in the unique position of being able to effectively mediate the relationship
2065 between users, identity providers, and websites. If the user agent can remove some of the risk and
2066 confusion associated with the typical authentication flows, users will be in a significantly
2067 better position than today.
2068
2069 A natural way to expose this information might be to extend the {{FederatedCredential}} interface

```

2016 with properties like authentication tokens, and possibly to add some form of manifest format with
2017 properties that declare the authentication type which the provider supports.
2018
2019 The API described here is designed to be extensible enough to support use cases that require user
2020 interaction, perhaps with websites other than the one which requested credentials. We hope that
2021 the Promise-based system we've settled on is extensible enough to support these kinds of
2022 asynchronous flows which could require some level of interaction between multiple browsing
2023 contexts (e.g. mediated activity on `idp.com` might resolve a Promise handed back to
2024 `rp.com`) or between devices and user agents (e.g. `[[WEBAUTHN]]`) in the future without
2025 redesigning the API from the ground up.
2026
2027 **Baby steps.**
2028 `</section>`
2029
2030

2070 with properties like authentication tokens, and possibly to add some form of manifest format with
2071 properties that declare the authentication type which the provider supports.
2072
2073 The API described here is designed to be extensible enough to support use cases that require user
2074 interaction, perhaps with websites other than the one which requested credentials. We hope that
2075 the Promise-based system we've settled on is extensible enough to support these kinds of
2076 asynchronous flows which could require some level of interaction between multiple browsing
2077 contexts (e.g. mediated activity on `idp.com` might resolve a Promise handed back to
2078 `rp.com`) or between devices and user agents (e.g. `[[WEBAUTHN]]`) in the future without
2079 redesigning the API from the ground up.
2080
2081 **Baby steps.**
2082 `</section>`
2083
2084