Credential Provider driven Windows Logon Experience

Updated for Windows® 10

Credential Provider driven Windows Logon Experience

Updated: 8/11/2016

Table of Contents

Requirements

Created in Windows Vista, Credential Providers collect credentials from users. They run in the LogonUI process as COM objects. To develop a Credential Provider you will need a Windows version at least Vista and the corresponding Windows SDK. You should also be reasonably familiar with programming Windows, be comfortable with Windows Security concepts, and be a reasonably experienced COM programmer.

Credential Providers may be installed on all Windows SKUs higher than Vista.

Introduction

This document describes how Windows Logon Experience is driven by Credential Provider Framework. It is intended to be read by developers and IT Professionals who wish to implement custom authentication mechanisms for Windows higher than Vista.

Before Windows Vista, organizations requiring custom authentication mechanisms for Windows logon were forced to replace the Microsoft Graphical Identification and Authentication Dynamic Link Library (MSGINA DLL) with their own GINA. In general, this architecture caused many problems for software vendors and IT professionals. Specifically, GINAs required constant upkeep and would routinely break with the release of each Service pack or new version of Windows. Another drawback of the GINA replacement model is that code written for authentication at Logon did not naturally extend to authentication in Credential UI.

In previous versions of Windows (such as Windows XP), MSGINA.DLL (or its replacement) was loaded after Winlogon detected a Secure Action Sequence (SAS) event. The user would enter their authentication credentials and then the GINA would pass this information back to Winlogon for authentication.

This limitation was removed in Windows Vista with the advent of the Windows Credential Provider Framework. In post-Vista era, when WinLogon wants to collect credentials, Logon UI queries each Credential Provider for the number of credentials it wishes to enumerate. Credential Providers have the option of specifying one of these tiles as the default. After all providers have enumerated their tiles, Logon UI displays them to the user. The user interacts with a tile to supply their credentials. Logon UI submits these credentials for authentication.

Combined with supporting hardware, Credential Providers can extend the Microsoft Windows operating system to enable users to logon through biometric (fingerprint, retinal, or voice recognition), password, PIN and Smart Card certificate, or any custom authentication package and schema a third party developer desires to create. Corporations and IT Professionals may develop and deploy custom authentication mechanisms for all domain users and may explicitly require users to use this custom logon mechanism.

Credential Providers are not enforcement mechanisms. They are used to gather and serialize credentials. The Local Authority and authentication packages enforce security.

Credential Providers may be designed to support Single Sign On (SSO), authenticating users to a secure network access point (leveraging RADIUS and other technologies) as well as machine logon. Credential Providers are also designed to support application-specific credential gathering, and may be used for

authentication to network resources, joining machines to a domain, or to provide administrator consent for User Account Control.

Multiple Credential Providers may co-exist on a machine.
Credential Providers are registered on a machine and responsible for:

- Describing the credential information required for authentication
- Handling communication and logic with external authentication authorities
- Packaging credentials for interactive and network logon

The Enums

These enumerations are used within Credential Providers to aid in code readability. They are used to communicate information between a Credential Provider and the process that instantiated it.

CREDENTIAL_PROVIDER_USAGE_SCENARIO

Your Credential Providers may be invoked in five separate scenarios. Credential Providers are not required to support all scenarios. A usage scenario is passed as a parameter in `SetUsageScenario`. Credential Providers store this parameter for reference throughout their lifecycle.
Providers may support creation on the Windows Logon screen and within Credential UI (this usage scenario includes the UAC prompt). Credential Provider authors may also define specific behavior when a machine is unlocked. If applicable, Credential Providers may also want to support changing private information used to identify the user (e.g. password or PIN). Some Credential Providers are designed to serve as Pre-Logon Access Providers (please refer to Appendix B for more information about PLAPs).

```
typedef enum _CREDENTIAL_PROVIDER_USAGE_SCENARIO
{
    CPUS_INVALID = 0,
    CPUS_LOGON,
    CPUS_UNLOCK_WORKSTATION,
    CPUS_CHANGE_PASSWORD,
    CPUS_CREDUI,
    CPUS_PLAP,
} CREDENTIAL_PROVIDER_USAGE_SCENARIO;
```

CPUS_INVALID
No usage scenario has been set for the Credential Provider. The scenario is not passed to `ICredentialProvider::SetUsageScenario`. If a Credential Provider stores its current usage scenario as a class member, this provides an initialization value before the first call to `SetUsageScenario`.

CPUS_LOGON
Implementing this usage allows the Credential Provider to enumerate tiles for workstation logon or unlock. Credential Providers implementing this usage scenario should be prepared to serialize credentials for authentication to the local authority.

CPUS_UNLOCK_WORKSTATION
Implementing this usage allows the Credential Provider to enumerate tiles for workstation unlock. Credential Providers implementing this usage scenario should be prepared to serialize credentials for authentication to the local authority.

Credential Providers implementing this usage scenario should enumerate the currently logged on user as the default tile. There are multiple ways to keep track which user is currently logged on to the system. The Credential Provider may maintain this information internally or leverage existing APIs (such as `WTSQuerySessionInformation`) to obtain it.

While `CPUS_LOGON` is the usage scenario at workstation logon, either `CPUS_LOGON` or `CPUS_UNLOCK_WORKSTATION` may be used during workstation unlock.

Several factors are used to determine the best usage scenario by the underlying Operating System and includes, but is not limited to:

- Operating system product type (e.g. `VER_NT_WORKSTATION`)
- Session type (Console or Remote)

Group policies such as:

- Hide entry points for Fast User Switching
- Interactive logon: Do not display last user name

CPUS_CHANGE_PASSWORD
This usage scenario allows the Credential Provider to enumerate tiles in response to a user request to change their password (or other private information such as a PIN). Credential Providers implementing this usage scenario should enumerate the currently logged on user as the default tile. This should not be implemented by Credential Providers that do not require some secret information (such as a password or PIN) that may be changed by the user. As with `CPUS_UNLOCK_WORKSTATION`, the Credential Provider may remember which user is currently logged on itself or leverage existing APIs (such as `WTSQuerySessionInformation`) to obtain this information.

CPUS_CREDUI
Implemeintg this usage scenario allows credentials serialized by the Credential Provider to be used for authentication on remote machines. Credential UI does not use the same instance of the provider as the Logon UI, Unlock Workstation, or Change Password. State information can not be maintained in the provider between instances of Credential UI. This usage scenario is also used for over the shoulder prompting in User Account Control.

CPUS_PLAP
Credential Providers responding to this usage scenario will be displayed on the Pre-Logon-Access Provider screen. (Please refer to Appendix B for more information about PLAPs)

CREDENTIAL_PROVIDER_FIELD_TYPE

Any Credential Providers that enable a user to authenticate with private information is useless without collecting the information. Credential Providers may offer Checkboxes, Combo boxes, editable text fields, and password (echoing dots) text fields for users to supply information. Credential Providers convey information to users through large text, small text, and images. Credential Providers may contain Hyper Links (although they are called Command Links).
Credential Providers do not actually draw their own UI. They specify elements to be displayed to the user within a usage scenario. The provider returns a set of fields (in various states) and the code instantiating the provider draws the fields as appropriate.

```
typedef enum _CREDENTIAL_PROVIDER_FIELD_TYPE
{
    CPFT_INVALID = 0,
    CPFT_LARGE_TEXT,
    CPFT_SMALL_TEXT,
    CPFT_COMMAND_LINK,
    CPFT_EDIT_TEXT,
    CPFT_PASSWORD_TEXT,
    CPFT_TILE_IMAGE,
    CPFT_CHECKBOX,
    CPFT_COMBOBOX,
    CPFT_SUBMIT_BUTTON,
} CREDENTIAL_PROVIDER_FIELD_TYPE;
```

CPFT_LARGE_TEXT
A standalone text label drawn in the larger of two font sizes.

CPFT_SMALL_TEXT
A standalone text label drawn in the smaller of two font sizes.

CPFT_COMMAND_LINK
An un-editable string a user may click to perform an action. The Credential Provider will be informed of the user's click and must do the appropriate action. (Refer to `CommandLinkClicked` for more information)

CPFT_EDIT_TEXT
This field is an edit box. Users may provide information for serialization by typing in this box.

CPFT_PASSWORD_TEXT
This box is similar in every way to `CPFT_EDIT_TEXT` with one notable exception. Characters typed in this box echo as dots on the user's screen.

CPFT_TILE_IMAGE
Prior to windows 10, a credential provider cannot specify more than one `CPFT_TILE_IMAGE` field, and it can only be used as the provider Logo icon. In Windows 10, we start allow tile image type as a regular credential field.

CPFT_CHECKBOX
A checkbox control that allows for checked and unchecked states.

CPFT_COMBOBOX
This control allows a user to select an option from a pre-defined set of discrete choices.

CPFT_SUBMIT_BUTTON
This field appears as a button on the credential tile. Pressing the button allows the user to submit their credentials. There may be one and only one `CPFT_SUBMIT_BUTTON` for any credential tile. Unlike Logon UI, which draws a special submit button in the tile layout, Credential UI hides this field and renders a single submit button for all credentials.

CREDENTIAL_PROVIDER_FIELD_STATE

Credential Provider fields may vary their behavior based on their state. Developers may choose to hide and display fields on the tile based on state. Recognized states are selected and de-selected. Fields may be displayed or hidden when the tile is selected, displayed or hidden when the tile is deselected, or may be displayed or hidden at all times regardless of tile state.

```
typedef enum _CREDENTIAL_PROVIDER_FIELD_STATE
{
    CPFS_HIDDEN = 0,
    CPFS_DISPLAY_IN_SELECTED_TILE,
    CPFS_DISPLAY_IN_DESELECTED_TILE,
    CPFS_DISPLAY_IN_BOTH,
} CREDENTIAL_PROVIDER_FIELD_STATE;
```

CPFS_HIDDEN
Do not show the control at all in any state. An example where this state could be used is a password field should only be shown after a thumb print was read. In this case, the field password field would begin in this state.

CPFS_DISPLAY_IN_SELECTED_TILE
Show the control in the credential when in the selected state.

CPFS_DISPLAY_IN_DESELECTED_TILE
Show the control in the credential when in the deselected state. This is the state in Logon UI/Credential UI that enumerates all of the valid credentials to be used.
In Windows 10, the Deselected state concept on LogonUI is retired, LogonUI only shows fields that is marked as "display in selected tile". Credential UI `(CPUS_CREDUI` scenario`)` still uses this though.

CPFS_DISPLAY_IN_BOTH
Show the control both when the credential tile is selected and when it is not selected.

CREDENTIAL_PROVIDER_FIELD_INTERACTIVE_STATE

Credential Provider tiles (Instances of `ICredentialProviderCredential`) may display their fields in a variety of states. The fields may be displayed as read only, disabled, or focused.

```
typedef enum _CREDENTIAL_PROVIDER_FIELD_INTERACTIVE_STATE
{
    CPFIS_NONE = 0,
    CPFIS_READONLY,
    CPFIS_DISABLED,
    CPFIS_FOCUSED,
} CREDENTIAL_PROVIDER_FIELD_INTERACTIVE_STATE;
```

CPFIS_NONE
The field is editable if allowed by the field type. Equivalent to `CPFIS_READONLY` for uneditable field types.

CPFIS_READONLY
This field interactive state is not used in Windows.

CPFIS_DISABLED
The field with this state will be disabled on LogonUI, user cannot interact with the control but can still see it. Note, it is only supported in post Windows 10.

CPFIS_FOCUSED
Credential Providers use this field interactive state to indicate the field should receive initial keyboard focus. This interactive state may not be specified for un-editable field types. If several editable fields specify state `CPFIS_FOCUSED`, the last of them (by their `dwIndex` order) will receive focus (first for prior Windows 10).

In Windows 10, field interactive state will be set during initial rendering or when the credential provider fires interactive state change events (e.g. user types 6 digits on the first field and CP moves the cursor focus to the second field), be careful when firing the events, because it may interrupt users.
Prior to Windows 10, only initial rendering respects focus state.

CREDENTIAL_PROVIDER_GET_SERIALIZATION_RESPONSE

This enumeration contains the values necessary for the ICredentialProviderCredential to communicate whether an attempt to serialize credentials completed successfully or not.

For instance, if a credential requires a PIN and answer to a secret question but only received the PIN then returning `CPGSR_NO_CREDENTIAL_NOT_FINISHED` signals the caller should allow the user to change their response. `CPGSR_NO_CREDENTIAL_FINISHED` means the caller should not attempt to serialize again and `CPGSR_RETURN_CREDENTIAL_FINISHED` implies serialization was successful.

```
typedef enum _CREDENTIAL_PROVIDER_GET_SERIALIZATION_RESPONSE
{
    CPGSR_NO_CREDENTIAL_NOT_FINISHED,
    CPGSR_NO_CREDENTIAL_FINISHED,
    CPGSR_RETURN_CREDENTIAL_FINISHED,
    CPGSR_RETURN_NO_CREDENTIAL_FINISHED,
} CREDENTIAL_PROVIDER_GET_SERIALIZATION_RESPONSE;
```

CPGSR_NO_CREDENTIAL_NOT_FINISHED

No credential was serialized because more information is needed.

CPGSR_NO_CREDENTIAL_FINISHED
This serialization response means that the Credential Provider has not serialized a credential but it has completed its work. This response has multiple meanings. It can mean that no credential was serialized and the user should not try again. This response can also mean no credential was submitted but the credential's work is complete. For instance, in the Change Password scenario, this response implies success.

CPGSR_RETURN_CREDENTIAL_FINISHED
A credential was serialized. This response implies a serialization structure was passed back.

CPGSR_RETURN_NO_CREDENTIAL_FINISHED

The credential provider has not serialized a credential, but has completed its work. The difference between this value and `CPGSR_NO_CREDENTIAL_FINISHED` is that this flag will force the logon UI to return, which will unadvise all the credential providers.

CREDENTIAL_PROVIDER_STATUS_ICON

When `ReportResult` is called on a credential, the credential may specify a status icon to display. For instance, if an incorrect password is entered the error icon can be returned. –This enumeration is only used in Logon UI. Note: Credential UI does not call ReportResult.

```
typedef enum _CREDENTIAL_PROVIDER_STATUS_ICON
{
    CPSI_NONE = 0,
    CPSI_ERROR,
    CPSI_WARNING,
    CPSI_SUCCESS,
} CREDENTIAL_PROVIDER_STATUS_ICON;
```

CPSI_NONE
Display no icon.

CPSI_ERROR
Display the error icon, deprecated in Windows 10.

CPSI_WARNING
Display the warning icon, deprecated in Windows 10.

CPSI_SUCCESS
Display the successful icon, deprecated in Windows 10.

The Structs

Two structs are defined within CredentialProvider.h. `CREDENTIAL_PROVIDER_FIELD_DESCRIPTOR` defines the format each field of a Credential Provider credential. `CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION` defines the serialization structure used within the Credential Provider Framework.

CREDENTIAL_PROVIDER_FIELD_DESCRIPTOR

Each UI element presented to the user on a tile is defined by the Credential Provider as a field. The Credential Provider describes these fields with a field descriptor. Field descriptors uniquely identify the field within the Credential Provider. Fields may not be added or subtracted from after they are defined for a particular usage scenario. Credential Providers must define all fields before enumerating tiles. Fields that will appear and disappear dynamically should be created before enumerating a tile. The Provider should hide and reveal the field using the `CPFS_HIDDEN`.

```
typedef struct _CREDENTIAL_PROVIDER_FIELD_DESCRIPTOR
{
    DWORD dwFieldID;
    CREDENTIAL_PROVIDER_FIELD_TYPE cpft;
    LPWSTR pszLabel;
    GUID guidFieldType;
} CREDENTIAL_PROVIDER_FIELD_DESCRIPTOR;
```

dwFieldID
This member specifies the unique identifier of the field. The identifier must be unique within the array of field descriptors returned by a given Credential Provider through `GetFieldDescriptorAt`. All fields should have a unique identifier regardless of whether they are displayed or hidden.

cpft
The type of the field from the list of possible fields in `CREDENTIAL_PROVIDER_FIELD_TYPE`.

pszLabel
This label names the field for accessibility technologies such as narrator. For instance, the standard password credential would have fields labeled “Username”, “Password”, and “Log On To”.

guidFieldType
A GUID that uniquely identifies a type of field. This member enables developers to wrap functionality provided by existing Credential Providers in their own provider. The guidFieldType allows these developers to specify a field by a unique value. Wrapping credential providers is not recommended as it can lead to problematic behavior that disables in box credential providers.

The Password Provider username, password, Smart Card Provider username, and PIN field each have a pre-defined `guidFieldType`.

Here are the guid types that Windows supports

Value	Meaning
CPFG_LOGON_USERNAME da15bbe8-954sd-4fd3-b0f4-1fb5b90b174b	The user name entered into a text box.
CPFG_LOGON_PASSWORD 60624cfa-a477-47b1-8a8e-3a4a19981827	The password entered into a text box.
CPFG_SMARTCARD_USERNAME 3e1ecf69-568c-4d96-9d59-46444174e2d6	The user name obtained from an inserted smart card.
CPFG_SMARTCARD_PIN 4fe5263b-9181-46c1-b0a4-9dedd4db7dea	The PIN obtained from an inserted smart card.
CPFG_CREDENTIAL_PROVIDER_LOGO 2d837775-f6cd-464e-a745-482fd0b47493	**Introduced in Windows 8**: The image used to represent a credential provider on the logon page.
CPFG_CREDENTIAL_PROVIDER_LABEL 286BBFF3-BAD4-438F-B007-79B7267C3D48	**Introduced in Windows 8**: The label associated with a credential provider on the logon page.

We recommend you to specify both CPFG_CREDENTIAL_PROVIDER_LOGO and CPFG_CREDENTIAL_PROVIDER_LABEL if you are developing a new credential provider. If they are not specified, Logon UI and Credential UI will use the first tile image field as logo and first text field as label.

CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION

Once the credential information has been gathered on the Credential tile, it must be packaged into a buffer. The final destination of serialized credentials depends on the usage scenario. In the Logon scenario, Logon UI will pass these serialized credentials to Winlogon. Ultimately these credentials are used by the LSA to authenticate the user for interactive logon. In the Credential UI scenario, the serialized buffer is passed back to the calling application. The calling application then uses the serialized buffer to authenticate.

The process of packaging credential information into a buffer is called “Serialization”. (For an example of Serialization, please refer to Appendix A)

The `CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION` structure defines the format all serialized credentials must take regardless of the authentication package used. After serialization, this structure will be passed back to Winlogon in the Logon UI case and back to the calling application in the Credential UI case. If you are using a custom authentication package, pass your credentials back in this structure the same way. Note that Credential Providers implementing `CPUS_LOGON` do not call `LsaLogonUser` – this call is handled by the system.

It is important to note that `clsidCredentialProvider` is used within Logon UI but is ignored by Credential UI. Credential UI throws this member away when it returns `cbSerialization` and `rgbSerialization` through the parameters passed in by reference from the caller. Credential UI passes `ulAuthenticationPacakage` to providers during `SetSerialization`.

Credential Providers may also choose to enumerate a credential tile if an input credential is received from `SetSerialization`. Within Credential UI this is useful if a user has supplied an incorrect user name or password. In this case, Credential UI will pass the invalid credentials back to the Credential Provider for re-enumeration so that the Credential Provider may present a partially pre-filled tile to the user in order to make it easier for the user to correct the information they supplied. Input credentials may take many forms. Credential Providers should be robust to receiving complete serialized credentials as well as partial credentials such as incomplete smart card certificates, domain credentials with no user name, etc. In many cases, an incomplete input credential is a hint about what kind of credential the caller wants (for example, this convention is used by callers who only wish to gather Smart Card credentials from the user). In `CPUS_LOGON`, `SetSerialization` is used to pre-fill information from a remote machine, such as in a Terminal Services connection. Logon UI will call `SetSerialization` zero or one times each enumeration cycle. If a Credential Provider Filter returns a credential serialization from `UpdateRemoteCredential`, Logon UI will pass that serialization as a parameter.

```
typedef struct _CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION
{
    ULONG   ulAuthenticationPackage;
    GUID    clsidCredentialProvider;
    ULONG   cbSerialization;
    [size_is(cbSerialization)] byte* rgbSerialization;
} CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION;
```

ulAuthenticationPackage

The unique identifier of an authentication package. This parameter is required when calling `LsaLogonUser`. In Credential UI this member is set before a `cpcs` is passed to the provider through `SetSerialization`. Credential Providers may examine this member in order to determine if they are able to return credentials for that authentication package. `ulAuthenticationPackage` is the same type of data that is returned from `LsaLookupAuthenticationPackage`.

clsidCredentialProvider
Credential Providers should assign their own Class ID to this member during serialization. Credential UI ignores this member.

cbSerialization
The size (in bytes) of the serialized credential contained in the memory pointed to by `rgbSerialization`.

***rgbSerialization**
This is an array of bytes containing serialized credential information. The exact format of this data depends on the authentication package targeted by a Credential Provider.

The Interfaces

In order to create a Credential Provider which works with Windows Vista and new versions of Windows, `ICredentialProvider` and `ICredentialProviderCredential` must be implemented.
`ICredentialProviderCredentialEvents` provides methods a Credential Provider may use to update its fields based on user activity. It is implemented by the host – either Logon UI or Credential UI.
Credential Provider authors who wish to prevent other Credential Providers on the machine from enumerating user tiles based on information unavailable before runtime should implement `ICredentialProviderFilter`.
`IConnectableCredentialProviderCredential` is used to enable tasks requiring pre-logon network connectivity. `IQueryContinueWithStatus` allows Logon UI to communicate user cancellation requests to a Credential Provider implementing `IConnectableCredentialProviderCredential` during the network connection process.
`ICredentialProviderEvents` provides a method allowing Credential Provider to notify Credential UI and Logon UI their Credentials have changed.
Implementations of `IQueryContinueWithStatus` and `ICredentialProviderCredentialEvents` are implemented by Microsoft. Credential Provider developers do not need to implement these interfaces.

ICredentialProvider

ICredentialProvider is one of the most important interfaces you will create when you author your Credential Provider. In some sense, this interface is the Credential Provider. Your Credential Provider interacts with Logon UI and Credential UI through this interface. It contains all the methods used in the setup and manipulation of a Credential Provider.

```
interface ICredentialProvider : IUnknown
{
    HRESULT SetUsageScenario([in] CREDENTIAL_PROVIDER_USAGE_SCENARIO cpus,
                             [in] DWORD dwFlags);
    HRESULT SetSerialization([in] const CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION* pcpcs);

    HRESULT Advise([in] ICredentialProviderEvents* pcpe, [in] UINT_PTR upAdviseContext);
    HRESULT UnAdvise();

    HRESULT GetFieldDescriptorCount([out] DWORD* pdwCount);
    HRESULT GetFieldDescriptorAt([in] DWORD dwIndex,
                                 [out] CREDENTIAL_PROVIDER_FIELD_DESCRIPTOR** ppcpfd);

    HRESULT GetCredentialCount([out] DWORD* pdwCount,
                               [out] DWORD* pdwDefault,
                               [out] BOOL* pbAutoLogonWithDefault);
    HRESULT GetCredentialAt([in] DWORD dwIndex,
                            [out] ICredentialProviderCredential** ppcpc);
};
```

SetUsageScenario

This required method allows the Credential Provider to be told how it will be used (for logon, unlock workstation, expired password change, etc). Credential Providers should return failure if the requested usage scenario is not supported. This usage scenario should affect the return values of `GetFieldDescriptorCount()` and `GetFieldDescriptorAt()` so that the appropriate fields are displayed depending on the scenario.

Throughout Logon UI's life, instantiated Credential Providers are maintained. Because of this, Logon UI can maintain state with regards to the Credential Provider. In particular, it remembers which provider and tile provided a credential blob. Logon UI is a very specialized application – it knows exactly where the credential blobs are destined and what types of error codes it can expect to see returned. In other words, it is possible to store state information in the provider for `CPUS_LOGON`, `CPUS_UNLOCK_WORKSTATION`, and `CPUS_CHANGE_PASSWORD`. The same is not true for Credential UI. Credential UI creates a new instance of the Provider every time an application calls `CredUIPromptForWindowsCredentials`. No state can be maintained in the provider between calls.

Credential Provider authors should also realize it is possible for serializations generated in one usage scenario to be used in a subsequent usage scenario. ICredentialProvider and ICredentialProviderCredential implementations must be robust enough to support this.

Return `S_OK` if `SetUsageScenario` completes successfully and the usage scenario is implemented. Return `E_NOTIMPL` if the call completes successfully but the scenario is unsupported. Credential Providers returning a failure status value are disabled for the specified scenario.

cpus

The value of this `CREDENTIAL_PROVIDER_USAGE_SCENARIO` indicates the scenario the Credential Provider has been created in.

dwFlags

These flags are set by Credential UI. Credential Providers should examine these flags when determining how many credentials to enumerate. For instance, if `CREDUIWIN_ADMINS_ONLY` is passed, Credential Providers should only enumerate credential tiles for the administrators on the local machine. For more information about these flags, please refer to the `CredUIPromptForWindowsCredentials` reference on msdn. Valid flags are:

```
CREDUIWIN_GENERIC
CREDUIWIN_CHECKBOX
CREDUIWIN_AUTHPACKAGE_ONLY
CREDUIWIN_IN_CRED_ONLY
CREDUIWIN_ENUMERATE_ADMINS
CREDUIWIN_ENUMERATE_CURRENT_USER
CREDUIWIN_PACK_32_WOW
```

SetSerialization

This required method accepts an input credential. Credential UI calls this method when an input credential has been supplied by an application (this credential could be partial or full). Logon UI calls `SetSerialization` when a filter returns a credential through `UpdateRemoteCredential`. It does not use this method when re-enumerating tiles after a call to `CredentialsChanged`. This method is always called after `SetUsageScenario`. Credential Providers that implement `CPUS_LOGON` and return a failure status value from this call will still be enabled.
Credential UI enforces the following rules:

- If `dwFlags` includes `CREDUIWIN_IN_CRED_ONLY`, all Credential Providers returning `S_OK` are enabled.

- If `dwFlags` includes `CREDUIWIN_AUTHPACKAGE_ONLY`, all Credential Providers returning a success status value will be enabled.
- If `dwFlags` includes neither of these flags, then Credential UI uses the same logic as Logon UI and all Credential Providers that implement `CPUS_CREDUI` will be enabled regardless of the status value they return.

pcpcs

The Credential Provider must examine `CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION` in the memory referenced by this parameter and determine if the serialization structure was passed in for display (i.e., a partial credential) or possible serialization (i.e., a full credential that should be auto-submitted). The values of this parameter's members (e.g., `ulAuthenticationPackage` and `rgbSerialization`) combined with the flags passed in `SetUsageScenario`, (assuming `CPUS_CREDUI` was passed), will determine how the provider will respond.

Credential Providers must verify the integrity and validity of input serializations. Credential UI and Logon UI do not perform any checks on this structure before passing it to the Credential Provider.
Note that in `CPUS_CREDUI` the value of `clsidCredentialProvider` will not be a valid Credential Provider. This field cannot be used to target a specific Credential Provider.

Advise

This is an optional method used to enable a Credential Providers to receive an `ICredentialProviderEvents` interface pointer from Logon UI and Credential UI during usage. This interface pointer allows asynchronous callback communication with Logon UI and Credential UI. If the `ICredentialProviderEvents` that is passed in by `Advise` is stored, it is the Provider's responsibility to call `AddRef` on it and `Release` it during `UnAdvise`. Credential Providers call `CredentialsChanged` through the interface pointer to initiate credential re-enumeration.

For instance, a Smart Card Provider would wish to re-enumerate its credentials when a smart card being inserted. Credential Providers that do not support asynchronous re-enumeration should return `E_NOTMIMPL`.

pcpe

This is a pointer to an object implementing `ICredentialProviderEvents`. The Credential Provider should use the `CredentialsChanged` method of this object to trigger Logon UI to re-initiate the `GetCredentialsCount` sequence of all Credential Providers installed on the machine. The Credential Provider is responsible for calling `AddRef()` on this pointer.

upAdviseContext

Each Credential Provider receives a unique cookie when Advise is called. This cookie is used when the Credential Provider makes subsequent calls to `ICredentialProviderEvents::CredentialsChanged`. This cookie allows Logon UI and Credential UI to correctly identify which Credential Provider has requested re-enumeration.

UnAdvise

Logon UI and Credential UI call `UnAdvise` to communicate the interface pointer is no longer valid. After receiving this call, the pointer to the `ICredentialProviderEvents` interface pointer is not longer valid. In some edge cases it is possible for an `UnAdvise` call to fail. Because it is less likely for Credential Provider's destructor to fail, Credential Providers should not use this method to clean up sensitive buffers. Sensitive buffers should be zeroed and freed within the Credential Provider's destructor.

GetFieldDescriptorCount

Required method which returns the field count of the UI fields required to display this Credential Provider's credentials. This field count is per usage scenario, but must be valid for the entire scenario. It is important for the Provider to return the count of all fields including those that are only displayed when selected, unselected, and those fields that are not yet displayed at all. This value cannot be changed until

a new `SetUsageScenario` call is made to the Provider or the `ICredentialProviderEvents` callback is used to force a re-enumeration. A Provider may show or hide fields in response to being selected or deselected, but all of the fields must be present.

***pdwCount**
The memory referenced by this parameter contains the number of field descriptors the Credential Provider contains.

GetFieldDescriptorAt
Required method which returns metadata describing one of the individual UI fields for displaying this Credential Provider's credentials. The index is zero based and is expected to be no greater than the count returned by `GetFieldDescriptorCount` minus one. It is the Provider ' s responsibility to make sure that the index is valid based on the number of fields they reported they would have and that the descriptor accurately describes the current state of the field (This will usually be just the defaults for the fields when the tiles are first enumerated).

***dwIndex**
This is the index of the field the field descriptor will be returned for.

****ppcpfd**
Credential Providers must allocate a `CREDENTIAL_PROVIDER_FIELD_DESCRIPTOR` structure with `CoTaskMemAlloc` and then set the field descriptor contents equal to appropriate values. This parameter allows a pointer to the field descriptor created by the Credential Provider to be returned to Logon UI and Credential UI.

GetCredentialCount
This is a required method used to communicate the number of credentials that can be enumerated through `GetCredentialAt`.

***pdwCount**
The memory referenced by this parameter contains the number of credentials the provider can return.

***pdwDefault**
The memory referenced by this parameter indicates the index of the credential the provider intends to treat as the default credential. A default credential is pre-selected when the Logon UI (or Credential UI) are ready for user interaction. As each Provider specifies a default credential tile, the following rules are applied in order determine if it will receive default focus or be used to autologon:

- If a default credential has already been specified and that credential is not intended to be used for autologon, AND this new default credential is intended to be used for autologon, then this new credential will be treated as the default.
- If this credential is from the last logged on provider, and there isn't already a default with autologon, then this new credential will be treated as the default.
- If no default credential has been specified yet, then this new credential will be treated as the default.

Credential Providers that do not wish to specify a default tile should return `CREDENTIAL_PROVIDER_NO_DEFAULT`.

***pbAutoLogonWithDefault**
The memory referenced by this parameter specifies whether the Credential Provider would like Logon UI or Credential UI to immediately call `GetSerialization` on the Provider's default tile.

GetCredentialAt
Required method which returns a specific credential from the Credential Provider. This is a zero based index into the number of credentials reported in `GetCredentialCount`. If the number of valid

credentials changes, the `ICredentialProviderEvents` given to the Provider in `Advise` should signal a request for a complete re-enumeration.

dwIndex
This parameter contains the index of the credential Logon UI or Credential UI is requesting.

***ppcpc*
This memory referenced by this parameter contains the instance of the credential returned to Logon UI and Credential UI. The Credential Provider allocates the memory for the credential and returns a pointer to it in this parameter.

ICredentialProviderEvents
This interface is implemented by Logon UI and Credential UI. It exposes a single method allowing Credential Providers to notify Credential UI and Logon UI their credentials have changed.

```
interface ICredentialProviderEvents : IUnknown
{
    HRESULT CredentialsChanged([in] UINT_PTR upAdviseContext);
};
```

CredentialsChanged
This is an asynchronous callback method Credential Providers use to signal to the host (Logon UI or Credential UI) the Credential Provider has changed its credential count, field count, or the specific indices within either of these sets.

upAdviseContext
The Credential Provider should pass back the interface pointer it received through `Advise` in this parameter.

ICredentialProviderCredential
This Interface defines the methods needed to implement a credential tile.

```
interface ICredentialProviderCredential : IUnknown
{
    HRESULT Advise([in] ICredentialProviderCredentialEvents* pcpce);
    HRESULT UnAdvise();

    HRESULT SetSelected([out] BOOL* pbAutoLogon);
    HRESULT SetDeselected();

    HRESULT GetFieldState([in] DWORD dwFieldID,
                          [out] CREDENTIAL_PROVIDER_FIELD_STATE* pcpfs,
                          [out] CREDENTIAL_PROVIDER_FIELD_INTERACTIVE_STATE* pcpfis);

    HRESULT GetStringValue([in] DWORD dwFieldID, [out, string] LPWSTR* ppsz);
    HRESULT GetBitmapValue([in] DWORD dwFieldID, [out] HBITMAP* phbmp);
    HRESULT GetCheckboxValue([in] DWORD dwFieldID, [out] BOOL* pbChecked, [out, string] LPWSTR*
ppszLabel);
    HRESULT GetSubmitButtonValue([in] DWORD dwFieldID, [out] DWORD* pdwAdjacentTo);

    HRESULT GetComboBoxValueCount([in] DWORD dwFieldID, [out] DWORD* pcItems, [out] DWORD*
pdwSelectedItem);
    HRESULT GetComboBoxValueAt([in] DWORD dwFieldID, DWORD dwItem, [out, string] LPWSTR*
ppszItem);

    HRESULT SetStringValue([in] DWORD dwFieldID, [in, string] LPCWSTR psz);
    HRESULT SetCheckboxValue([in] DWORD dwFieldID, [in] BOOL bChecked);
    HRESULT SetComboBoxSelectedValue([in] DWORD dwFieldID, [in] DWORD dwSelectedItem);
    HRESULT CommandLinkClicked([in] DWORD dwFieldID);

    HRESULT GetSerialization([out] CREDENTIAL_PROVIDER_GET_SERIALIZATION_RESPONSE* pcpgsr,
                             [out] CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION* pcpcs,
                             [out] LPWSTR* ppszOptionalStatusText,
```

```
                              [out] CREDENTIAL_PROVIDER_STATUS_ICON* pcpsiOptionalStatusIcon);
    HRESULT ReportResult([in] NTSTATUS ntsStatus,
                         [in] NTSTATUS ntsSubstatus,
                         [out] LPWSTR* ppszOptionalStatusText,
                         [out] CREDENTIAL_PROVIDER_STATUS_ICON* pcpsiOptionalStatusIcon);
};
```

Advise
Logon UI and Credential UI use this method to pass the Credential a pointer to an instance of `ICredentialProviderCredentialEvents` enabling synchronous callback communication. `Advise` and `UnAdvise` are called in pairs and are UI blocking. Return `E_NOTIMPL` if you do not need to implement this method.

*pcpce
The instance of `ICredentialProviderCredentialEvents` this pointer references allows the Credential Provider to request that Logon UI or Credential UI update fields on a tile. The Credential Provider is responsible for calling `AddRef()` on this pointer.

UnAdvise
`UnAdvise` and `Advise` are called in pairs. Credential UI and Logon UI call `UnAdvise` when the pointer `*pcpce` supplied in `Advise` is no longer valid.

SetSelected
This method is called when a credential is selected. Returning any value other than `S_OK` causes Credential UI and Logon UI to behave as if no selection occurred.

**pbAutoLogon*
The contents of `pbAutoLogon` can be set by the credential to indicate whether or not the selecting of the credential means it should attempt to logon immediately. For instance, a Credential Provider enumerating an account without a password may want to return this as true.

In Windows 10, if a credential provider wants to autologon the user where we think may be inappropriate, we will paint a “sign in” button as a speed bump. For example, when a user with empty password locks or signs out we don’t directly log the user back in.

SetDeselected
This method is called when a credential loses selection. Credential Providers should use this method to purge all buffers containing sensitive information (for instance, a password or PIN) in addition to purging them in the destructor of the class that stores them.

GetFieldState
This method allows Logon UI and Credential UI to retrieve information about a particular field of a credential in order to display this information in the user tile.

dwFieldID
This is the ID of the field within the Credential Provider. Fields within a Credential Provider are not required to be ordered in any way.

*pcpfs
This pointer to a `CREDENTIAL_PROVIDER_FIELD_STATE` communicates when Credential UI and Logon UI should show the field on the tile. For instance, the empty Password Provider tile does not display the user name and password fields when a tile is not selected.

*pcpfis
This pointer to a `CREDENTIAL_PROVIDER_FIELD_INTERACTIVE_STATE` communicates the state of the field to Credential UI and Logon UI.

GetStringValue
Credential UI and Logon UI use this method to obtain the `pszLabel` for a field. This is used to obtain values for `CPFT_LARGE_TEXT`, `CPFT_SMALL_TEXT`, labels of `CPFT_COMMAND_LINK` labels, `CPFT_EDIT_TEXT`, and `CPFT_PASSWORD_TEXT`.

dwFieldID
This parameter specifies the Credential Provider field a string is needed for.

*ppsz
The memory referenced by this parameter contains the string returned to Logon UI and Credential UI.

GetBitmapValue
Credential UI and Logon UI obtain a bitmap to display in the user tile by calling this method. Ownership of the bitmap should be given to the credential provider framework after calling this API. The caller should not destroy the bitmap as the framework is responsible for freeing the bitmap handle.

dwFieldID
This parameter specifies the Credential Provider field a user tile bitmap is needed for.

*phbmp
The memory referenced by this parameter contains the bitmap returned to Logon UI and Credential UI.

GetCheckboxValue
Credential UI and Logon UI use this method to obtain

dwFieldID
This parameter specifies the Credential Provider field a checkbox value is needed for.

*pbChecked
The boolean referenced by this parameter indicates the state of the checkbox to Logon UI and Credential UI.

*ppszLabel
The memory referenced by this parameter contains returns the label of the checkbox returned to Logon UI and Credential UI.

GetSubmitButtonValue
Logon UI uses this method to obtain information about which field the submit button should be placed after. Credential UI never calls this method.

dwFieldID
This parameter specifies the Credential Provider field a submit button value is needed for.

*pdwAdjacentTo
This parameter contains the field ID the submit button is adjacent to.

Hiding a submit button is not recommended unless your credential provider always performs auto-submit, because it may be hard for touch screen users to find a way to submit credentials.

GetComboBoxValueCount
Credential UI and Logon UI use this method to obtain the number of items in a combo box and which item should have initial selection.

dwFieldID
This parameter specifies the Credential Provider field a combo box value count is needed for.

*pcItems
The memory referenced by this parameter contains the number of items in the combo box.

*pdwSelectedItem
This parameter contains the value of the combo box item which will be selected.

GetComboBoxValueAt
Credential UI and Logon UI use this method to obtain the value of a string in the combo box at the index specified by the caller.

dwFieldID
This parameter specifies the Credential Provider field corresponding to the combo box an item will be returned from.

dwItem
This is the index of the item desired.

*ppszItem
The Credential Provider credential returns the appropriate string in the memory referenced by this parameter.

SetStringValue
Credential UI and Logon UI use this method to set the value of strings on `CPFT_EDIT_TEXT` fields as the user types in them.

dwFieldID
This parameter specifies the Credential Provider field where a string value will be set.

Psz
The new value of the field.

SetCheckboxValue
Credential UI and Logon UI use this method to indicate to the credential that a checkbox value has been changed (set to true or false).

dwFieldID
This parameter specifies the Credential Provider field a check box value will be set.

bChecked
This parameter indicates the value to set the checkbox to – true is checked, false is unchecked.

SetComboBoxSelectedValue
Credential UI and Logon UI use this method to indicate to the credential that a Combo Box item has been selected.

dwFieldID
This parameter specifies the Credential Provider field where a Combo Box item is selected.

dwSelectedItem
This value of this parameter indicates the item selected.

CommandLinkClicked
Credential UI and Logon UI use this method to indicate to the credential that a Command Link was clicked.

dwFieldID
This parameter specifies the Credential Provider field of the Command Link that was clicked.

GetSerialization
This method is required. It is expected to take appropriate action based on the usage scenario. In CPUS_LOGON and CPUS_UNLOCK_WORKSTATION, the information credential information should be packed up into a binary stream for transmission to Winlogon and eventual submission to LSA. In CPUS_CREDUI, the information should be serialized for delivery to the calling application.

In CPUS_CHANGE_PASSWORD, no actual credential serialization takes place. Instead, the Credential should take appropriate actions to update the users secret information and return CPGSR_NO_CREDENTIAL_FINISHED in pcpgsr.

*pcpgsr
The Credential uses this parameter to indicate the success or failure of the attempts to Serialize Credentials. Refer to CREDENTIAL_PROVIDER_GET_SERIALIZATION_RESPONSE above for more information.

*pcpcs
This is a pointer to the CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION structure (if any) serialized by the Credential.

*ppszOptionalStatusText
This parameter may be used to send additional information back to Logon UI that will be displayed after serialization.

*pcpsiOptionalStatusIcon
This parameter may be used to specify a CREDENTIAL_PROVIDER_STATUS_ICON. This icon will be shown on the Credential after the call to GetSerialization returns.

ReportResult
This method is called by Logon UI to allow the Credential to format the ntstatus code returned after the last logon attempt. Credential UI never calls this method. Credential Provider authors use this method to provide meaningful error and success messages to the user. The status text returned from this call will appear on the selected credential.

ntStatus
Logon UI passes the ntStatus returned by Winlogon from LSA in this parameter.

ntSubstatus
Logon UI passes the ntSubstatus returned by Winlogon from LSA in this parameter.

*ppszOptionalStatusText
This parameter is used to return the error message that will be displayed to the user.

pcpsiOptionalStatusIcon
This parameter may be used to specify a CREDENTIAL_PROVIDER_STATUS_ICON. This icon will be shown on the Credential after the call to ReportResult returns.

ICredentialProviderCredential2

In Windows 8, we introduced “User First” concept to provide personalized logon experience (Details see Credential Provider Framework Changes in Windows 8.docx). In other to create a V2 credential provider, developers need to implement ICredentialProviderCredential2, basically telling LogonUI which user has what sign-in options.

If you are considering writing a new credential provider, V2 is what we recommend. For more details, please see Appendix B: Creating a V2 Credential Provider

```
interface ICredentialProviderCredential2 : ICredentialProviderCredential
{
```

```
    HRESULT GetUserSid([out, string, annotation("_Outptr_result_maybenull_")] LPWSTR *sid);
};
```

ICredentialProviderCredentialEvents

This interface is implemented by Microsoft. Credential Providers do not need to implement this interface. Objects implementing this interface contain methods credentials may call in order to change the state of their UI.

The methods exposed by this interface are only intended to be called by the selected credential passing `this` as the first parameter. Behavior is undefined if you pass a credential other than the one `Advise` was called on. Credential Providers should not pass this object to other threads. If a Credential Provider has information on another thread that it wants sent to Logon UI or Credential UI through this interface, it should communicate between its threads such that the thread that received the Advise is the thread that calls into the interface.

FieldIDs do not need to be sequential but they must be unique within a Credential.

```
interface ICredentialProviderCredentialEvents : IUnknown
{
    HRESULT SetFieldState([in] ICredentialProviderCredential* pcpc,
                          [in] DWORD dwFieldID,
                          [in] CREDENTIAL_PROVIDER_FIELD_STATE cpfs);

    HRESULT SetFieldInteractiveState([in] ICredentialProviderCredential* pcpc,
                                     [in] DWORD dwFieldID,
                                     [in] CREDENTIAL_PROVIDER_FIELD_INTERACTIVE_STATE cpfis);

    HRESULT SetFieldString([in] ICredentialProviderCredential* pcpc,
                           [in] DWORD dwFieldID,
                           [in, string, unique] LPCWSTR psz);

    HRESULT SetFieldCheckbox([in] ICredentialProviderCredential* pcpc,
                             [in] DWORD dwFieldID,
                             [in] BOOL bChecked,
                             [in] LPCWSTR pszLabel);

    HRESULT SetFieldBitmap([in] ICredentialProviderCredential* pcpc,
                           [in] DWORD dwFieldID,
                           [in] HBITMAP hbmp);

    HRESULT SetFieldComboBoxSelectedItem([in] ICredentialProviderCredential* pcpc,
                                         [in] DWORD dwFieldID,
                                         [in] DWORD dwSelectedItem);

    HRESULT DeleteFieldComboBoxItem([in] ICredentialProviderCredential* pcpc,
                                    [in] DWORD dwFieldID,
                                    [in] DWORD dwItem);

    HRESULT AppendFieldComboBoxItem([in] ICredentialProviderCredential* pcpc,
                                    [in] DWORD dwFieldID,
                                    [in, string] LPCWSTR pszItem);

    HRESULT SetFieldSubmitButton([in] ICredentialProviderCredential* pcpc,
                                 [in] DWORD dwFieldID,
                                 [in] DWORD dwAdjacentTo);

    HRESULT OnCreatingWindow([out] HWND* phwndOwner);
};
```

SetFieldState

A field state can be updated with this callback after `Advise` is called on the credential.

*pcpc

A pointer to the credential containing the field whose field state is being set.

dwFieldID
This is the FieldID of the field whose state is being set.

cpfs
The `CREDENTIAL_PROVIDER_FIELD_STATE` the field will be set to.

`SetFieldInteractiveState`
A field interactive state can be updated with this callback after `Advise` is called on the credential.

***pcpc**
A pointer to the credential containing the field whose field interactive state is being set.

dwFieldID
This is the FieldID of the field whose interactive state is being set.

Cpfis
The `CREDENTIAL_PROVIDER_FIELD_INTERACTIVE_STATE` the field will be set to.

`SetFieldString`
This callback allows credentials to update strings on their member fields after an `Advise` is called.

***pcpc**
A pointer to the credential containing the field whose string is being set.

dwFieldID
This is the FieldID of the field whose string is being set.

psz
This is the value the credential's string will be set to.

`SetFieldCheckbox`
This callback allows credentials to update their checkbox fields after `Advise` is called.

***pcpc,**
A pointer to the credential containing the field whose checkbox value is being set.

dwFieldID
This is the FieldID of the checkbox field being set.

bChecked
This parameter sets the value of the checkbox. True corresponds to checked and false corresponds to unchecked.

pszLabel
This string allows the caller to set the value of the checkbox label.

`SetFieldBitmap`
This callback allows credentials to update their user tile image field after `Advise` is called.

***pcpc**
A pointer to the credential whose user tile image is being set.

dwFieldID
This is the FieldID of the credential's user tile image.

hbmp
The user tile image's value will be set to this parameter.

SetFieldComboBoxSelectedItem
This callback allows credentials to update the selected item of a member combo box after `Advise` is called.

***pcpc**
A pointer to the credential whose member combo box selection is being set

dwFieldID
This is the FieldID of the credential's combo box whose selection is being set.

dwSelectedItem
This is the index of the item that will be selected within the combo box.

DeleteFieldComboBoxItem
This callback allows credentials to delete an item from their member combo box after `Advise` is called.

***pcpc**
A pointer to the credential whose member combo box contains an item to be deleted.

dwFieldID
This is the FieldID of the credential's combo box containing an item to delete.

dwItem
This is the item that will be deleted within the combo box.

AppendFieldComboBoxItem
This callback allows credentials to add an item to a member combo box after `Advise` is called.

***pcpc**
A pointer to the credential containing a combo box where an item will be added.

dwFieldID
This is the FieldID of the credential's combo box where an item will be added.

pszItem
The value of this string will be added to the combo box.

SetFieldSubmitButton
This callback allows credentials to set the field their submit button appears adjacent to after `Advise` is called.

***pcpc**
A pointer to the credential whose submit button location is being set.

dwFieldID
This is the FieldID of the credential's submit button field.

dwAdjacentTo
This is the FieldID of the field the submit will appear next to after this method completes.

OnCreatingWindow
This callback allows credentials to retrieve their parent's hwnd after `Advise` is called. This HWND is useful for parenting dialogs such as a messagebox. It is mandatory to parent your dialog to the HWND provided by this call. Credential UI and Logon UI cannot cancel the dialog in the event of a timeout if the dialog is not parented properly.

Dialogs in Logon UI that receive no input for two minutes will be cancelled automatically. Dialogs presented during calls to `IConnectableCredentialProviderCredential::Connect` on the PLAP screen will never be cancelled due to inactivity.

Credential Providers should follow Windows Vista UX guidelines and use the controls provided by within the Credential Provider framework to show all UI. If a specific scenario requires additional UI, please review Windows Vista UX guidelines when authoring your dialog.

*phwndOwner

This parameter is used to return the HWND of the appropriate parent window.

ICredentialProviderCredentialEvents2

In Windows 8, we introduced batch field update mechanism.

```
interface ICredentialProviderCredentialEvents2 : ICredentialProviderCredentialEvents
{
    HRESULT BeginFieldUpdates();
    HRESULT EndFieldUpdates();
    HRESULT SetFieldOptions([in] ICredentialProviderCredential *credential,
                            [in] DWORD fieldID,
                            [in] CREDENTIAL_PROVIDER_CREDENTIAL_FIELD_OPTIONS options);
};
```

ICredentialProviderFilter

Third party credential providers should not use this interface and should not filter out or disable system credential providers on a desktop. If an enterprise deploys a third party credential provider and wants to disable system credential providers that is a decision a domain administrator should make carefully. System policies exist that allow administrators to filter out credential providers and should be used instead of building filters directly into a third party credential provider.

IConnectableCredentialProviderCredential

Any Credential Provider designed to connect to the network must implement this interface. All long running tasks such as connecting to a network should be handled within the interface's `Connect` method.

```
interface IConnectableCredentialProviderCredential : ICredentialProviderCredential
{
    HRESULT Connect([in] IQueryContinueWithStatus* pqcws);
    HRESULT Disconnect();
};
```

Connect

`Connect` is called after the user clicks the Submit button within the PLAP logon screen. Connect is called before `GetSerialization`. When Logon UI calls `Connect`, it passes the method a pointer to an `IQueryContinueWithStatus`.

*pqcws

This is a pointer to instance of `IQueryContinueWithStatus` passed by Logon UI.

Disconnect

After a successful call to `Connect`, Logon UI displays a Disconnect button to the user. If the user clicks Disconnect, Logon UI calls `Disconnect` on every Credential Provider that implements `IConnectableCredentialProviderCredential`.

IQueryContinueWithStatus

Credential Providers use `IQueryContinueWithStatus` for two purposes. They may call `QueryContinue` to during their attempts to connect to the network in order to determine if they should continue these attempts. They may also use `SetStatusMessage` to display messages to user while attempting to establish a network connection.

```
interface IQueryContinueWithStatus : IQueryContinue
{
    HRESULT SetStatusMessage([in, string] LPCWSTR psz);
};
```

SetStatusMessage
This method allows the credential to pass status messages back to Logon UI. These messages are displayed to the user by Logon UI. During lengthy attempts to `Connect` this is especially useful.

Psz
This value of the memory referenced by this member will be displayed to the user by Logon UI.

QueryContinue
A simple example of a useful event this function communicates to the Credential Provider is a user clicking Cancel before a connection is established. A well behaved Credential Provider will call this function periodically during its attempt to connect to the network.

Appendix A – Serialization

Example of a serialized credential

This is an example of a serialized credential from the Password Provider. The credentials entered were SAMPLEDOMAIN\SAMPLEUSERNAME and the password entered was SAMPLEPASSWORD. Notice the structure of the information and that the password is encrypted. Your Credential Provider should encrypt all secret information (such as passwords) during serialization. You may protect your credentials using `CredProtect`

```
00000000 02 00 00 00 18 00 18 00 24 00 00 00 1C 00 1C 00 ........$.......
00000010 3C 00 00 00 9C 00 9C 00 58 00 00 00 00 00 00 00 <.......X.......
00000020 00 00 00 00 53 00 41 00 4D 00 50 00 4C 00 45 00 ....S.A.M.P.L.E.
00000030 44 00 4F 00 4D 00 41 00 49 00 4E 00 53 00 41 00 D.O.M.A.I.N.S.A.
00000040 4D 00 50 00 4C 00 45 00 55 00 53 00 45 00 52 00 M.P.L.E.U.S.E.R.
00000050 4E 00 41 00 4D 00 45 00 40 00 40 00 44 00 07 00 N.A.M.E.@.@.D...
00000060 08 00 0C 00 0A 00 0D 00 77 00 41 00 41 00 41 00 ........w.A.A.A.
00000070 41 00 41 00 64 00 53 00 6A 00 57 00 41 00 41 00 A.A.d.S.j.W.A.A.
00000080 41 00 41 00 41 00 41 00 67 00 36 00 6F 00 66 00 A.A.A.A.g.6.o.f.
00000090 39 00 63 00 58 00 42 00 6E 00 67 00 70 00 4B 00 9.c.X.B.n.g.p.K.
000000A0 38 00 53 00 61 00 61 00 77 00 4E 00 6A 00 72 00 8.S.a.a.w.N.j.r.
000000B0 48 00 77 00 45 00 56 00 76 00 6C 00 44 00 59 00 H.w.E.V.v.l.D.Y.
000000C0 53 00 4C 00 47 00 69 00 48 00 68 00 71 00 77 00 S.L.G.i.H.h.q.w.
000000D0 6C 00 47 00 55 00 46 00 64 00 4F 00 70 00 55 00 l.G.U.F.d.O.p.U.
000000E0 57 00 47 00 69 00 69 00 78 00 4B 00 71 00 6F 00 W.G.i.i.x.K.q.o.
000000F0 6C 00 57 00                                     l.W
```

For more information on the structure of serialized credentials, please refer to http://msdn.microsoft.com/msdnmag/issues/05/06/SecurityBriefs/ .

Appendix B – Creating a V2 Credential Provider

The following sections provide a guide for both the required and optional steps needed to implement a v2 credential provider.

Implementing ICredentialProviderCredential2

A valid SID must be returned by the `GetUserSid` function.
In this context, “valid” means that the returned SID must be one of the users currently enumerated by Logon UI. A credential provider can implement the `ICredentialProviderSetUserArray` interface to get the `ICredentialProviderUserArray` object from `ICredentialProviderSetUserArray::SetUserArray`, which allows the credential provider to know which users are enumerated by Logon UI. Logon UI will call `ICredentialProviderSetUserArray::SetUserArray` before `ICredentialProvider::GetCredentialCount` so that the `ICredentialProviderUserArray` object is ready when a credential provider is about to return credentials.

The `ICredentialProviderUserArray` object contains a list of `ICredentialProviderUser` objects, which can be queried to gather information about each user Logon UI is enumerating. For example, a user's SID or username may be gathered by using `ICredentialProviderUser::GetStringValue(PKEY_Identity_PrimarySid)` and `ICredentialProvider::GetStringValue(PKEY_Identity_UserName)`, respectively. The `ICredentialProviderUserArray ::SetProviderFilter` method can also be used to further filter this list down to either all of the Microsoft accounts present or the combination of local and domain accounts. While this is optional, using the user array is very helpful for v2 credential providers because it allows them to walk through all of the enumerated users and determine which user they want to associate with.

The following property keys are available on an `ICredentialProviderUser` object for a credential provider to query. For each key, an example is provided based on a user with the following properties:

- Domain user
    - Domain: contoso
    - User name: lisa
    - Friendly name: Lisa Andrews
- Local user:
    - PC name: lisa-pc
    - User name: lisa
    - Friendly name: Lisa Andrews
- Microsoft account:
    - Email address: lisa@contoso.com
    - Friendly name: Lisa Andrews

Property Key	**Description**	**Examples**
PKEY_Identity_UserName	Returns the user name.	Local user: “Lisa” Domain user: ”contoso\lisa” Microsoft account: ”lisa@contoso.com”
PKEY_Identity_QualifiedUserName	Returns the name that can be used to pack an authentication buffer.	Local user: “lisa-pc\lisa” Domain user: “contoso\lisa” Microsoft account: “<account provider name>\lisa@contoso.com”
PKEY_Identity_DisplayName	Returns the user's friendly name.	Local user: “Lisa Andrews” Domain user: “Lisa Andrews” Microsoft account: “Lisa Andrews”
PKEY_Identity_LogonStatusString	Returns a localized string that indicates the user’s logon status.	“Locked” “Signed-in” “Remotely signed-in from <PC name>”
PKEY_Identity_PrimarySid	Returns the user’s SID.	{S-1-5-21-2279990834-2601404236-735077814-1001}
PKEY_Identity_ProviderID	Returns the user’s provider ID, which can be used to tell if a user	Local user: {A198529B-730F-4089-B646-

	is a Microsoft account user.	A12557F5665E} Domain user: {A198529B-730F-4089-B646-A12557F5665E} Microsoft account: Not pre-defined

The following code shows how ICredentialProviderCredential2::GetUserSid should be implemented: (Please note, LogonStatus String is only available in windows 8/8.1)

```
// Gets the SID of the user corresponding to the credential.
HRESULT CSampleCredential::GetUserSid(__deref_out PWSTR *ppszSid)
{
    *ppszSid = nullptr;
    HRESULT hr = E_UNEXPECTED;

    // _pszUserSid is a private member of CSampleCredential
    if (_pszUserSid != nullptr)
    {
        // ppszSid will be freed by Logon UI
        hr = SHStrDupW(_pszUserSid, ppszSid);
    }
    // Return S_FALSE with a null SID in ppszSid for the
    // credential to be associated with an anonymous user tile.
    else if (_fIsOtherUserTile)
    {
        hr = S_FALSE;
    }

    return hr;
}
```

Logon UI will use the returned SID to associate the credential tile with a user tile. If the SID for user “Lisa” is returned, the credential tile will appear as an option when Lisa is selected.
If you want to enumerate an anonymous tile (i.e. a tile that is not associated with a particular user), return a **`null`** SID and **`S_FALSE`**. This will cause the tile to appear when the “Other user” tile is selected on a domain-joined PC. If the PC is not joined to a domain, this tile will not appear in normal scenarios.

Specify an icon to represent the credential provider

v2 credential providers are represented by icons that are displayed underneath the “Sign-in options” link. These icons are specified by the provider as explained in this procedure:

1. Define a CPFT_TILE_IMAGE field in the credential.
2. In the field descriptor, the `guidFieldType` must be set to the following GUID to indicate that the tile image field is an icon: `CPFG_CREDENTIAL_PROVIDER_LOGO`

The recommended size for this image is 72x72 px, since the image may be scaled based on a user’s display settings. This size allows for effective scaling to both larger and smaller sizes in Logon UI.

Specify a text label for the provider

In addition to the icon, a v2 credential provider also provides a text string to state what it is. This string appears in a pop-up when a user hovers over the credential provider’s icon underneath “Sign-in options”.

To do this, define a separate `CPFT_SMALL_TEXT` string field in your credential and set the `guidFieldType` to `CPFG_CREDENTIAL_PROVIDER_LABEL`. Doing so allows a credential provider to show different labels if more than one credential is associated with a user.
Since this string will supplement the icon in representing the credential provider to the user, it should be descriptive enough that users understand what it is. For example, the inbox password provider's description is "Password" and the picture password provider's description is "Picture Password."

Do not select a credential provider tile by default

As described previously, Logon UI, selects the appropriate user on Logon UI and, for that user, the appropriate credential provider.
As such, a provider should not select a default tile. Doing so will break the designed model of Logon UI, leading to user confusion. In addition, with Logon UI's new selection logic, doing this should be unnecessary since the user will already have the most appropriate provider selected by default.

Do not show the user name, signed-in status, or tile image

All of these items are now handled by Logon UI without any additional work needed by the credential provider.

However, a user name input field is still required for an anonymous tile.

Appendix C – Pre-Logon Access Providers

Overview

A Pre-Logon Access Provider (PLAP) is a special type of Credential Provider (CP) that allows users to make a network connection before logging into their machine.

An Example

Windows Vista ships with a single PLAP installed by default. If you're reading this document and are not familiar with PLAPs, you may find it helpful to follow this example in order to gain a basic understanding of where these special CPs fit within the Windows Vista landscape.
Note: You will need an access to a VPN to complete this example.

- Log on to your Windows machine using an account with Administrator privileges
- Navigate to the **Set up a connection or network** dialog
    - You can reach this location by opening the Control Panel and searching for "VPN" in the search box located in the upper right corner of the window
- Click **Set up a virtual private network (VPN) connection**
    - Fill in all necessary information for your VPN
    - Check **Don't connect now, just set it up**
    - After the test connection is successful complete setup by pressing the **close** button
- Log off
- In the lower right corner of the Logon screen you should now see the new PLAP button to the left of the power button
- Click the PLAP button
- Select the VPN PLAP tile you just created
- Enter your VPN credentials
- Click Submit button
- After connection to the VPN is complete
- Return to the logon screen
- Type your Windows Credentials
- Click Submit button
- When your desktop is available, hover over the network connection icon in the system tray
- Notice you are now connected to the VPN

Supported Scenarios

PLAPs are intended to be used in the following scenarios

- Network authentication and machine logon are handled by separate CPs. Slight variants of this scenario include
    - A user has the option of connecting to a network (such as connecting to a VPN in the example above) before logging in to the machine but is not required to make this connection.
    - Network authentication is required in order to retrieve information used during interactive authentication on the local machine.
    - Multiple network authentications are followed by one of the other scenarios. For example, a user authenticates to an ISP, then authenticates to a VPN, and then uses their user account credentials to log in locally.
    - Cached credentials are disabled and a RAS/VPN connection is required before logon to authenticate the user
    - A domain user does not have a local account set up on a domain joined machine and must establish a RAS/VPN connection before completing interactive logon
- Network authentication and machine logon are handled by the same Credential Provider and the user is required to connect to the network before logging on to the machine.

Scenarios where users will authenticate to a network and logon to the machine using separate Credential Providers require a Credential Provider that implements `CPUS_PLAP`. The Credential Provider that handles network authentication will be registered as a PLAP rather than a normal Credential Provider. Your provider will not enumerate tiles on Logon UI. Users will see them only after clicking the PLAP button.
When network authentication and local interactive authentication are handled by the same Credential Provider, your Credential Provider may be implemented and registered as normal. In this case developers should implement `IConnectableCredentialProviderCredential`. Your Single Sign (SSO) on provider will handle network authentication in `Connect` and then provide the credentials to authenticate to the local machine during `GetSerialization`.
Single Sign On Providers (SSOs) may be developed as PLAPs or standard Credential Providers. The decision depends on the particular scenario the solution addresses. If your scenario requires allocating an indeterminate amount of time for the SSO to connect, you will want to implement your solution as a PLAP. Logon UI expects calls to `Connect` in `CPUS_LOGON` usage scenario to return within a reasonable amount of time. Logon UI does not restrict the amount of time that calls to `Connect` in the `CPUS_PLAP` usage scenario may take to complete.
SSOs implemented as PLAPs will only be accessible to users after they press the network logon button to reveal the PLAPs installed on the machine.

Required interfaces

PLAPs must implement `ICredentialProvider` and `ICredentialProviderCredential` (or `IConnectableCredentialProviderCredential`). The provider must register itself in the registry like any other Credential Provider. PLAPS must meet two additional requirements

1) The PLAP must not return an error when `CPUS_PLAP` is passed to its implementation of `ICredentialProvider::SetUsageScenario`
2) PLAPs must register under `HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Authentication¥PLAP Providers`

IConnectableCredentialProviderCredential

Any Credential Provider designed to connect to the network must implement this interface. All long running tasks such as connecting to a network should be handled within the interface's `Connect` method. All PLAPs must implement this interface.

Connecting to the Network

By definition, implementing `IConnectableCredentialProviderCredential` requires two methods to support network operations - - `Connect` and `Disconnect`. `Connect` is called after the user clicks the Submit button within the PLAP logon screen. When Logon calls `Connect`, it passes the method a specific implementation of `IQueryContinueWithStatus`.
`IQueryContinueWithStatus` is used to communicate a message to the user while connecting. Logon UI receives messages through our implementation of this interface.
PLAP authors do not need to implement `IQueryContinueWithStatus` since this mechanism is specific to Logon UI and is passed into the PLAP through `Connect`.

Appendix D – Wrapping In-box Credential Providers

Wrapping these credential providers is not supported:
FaceCredentialProvider
IrisCredentialProvider
PicturePasswordLogonProvider
PinLogonProvider
WinBioCredentialProvider

Wrapping credential providers supplied by the operating system that are not in this list is supported but not recommended. It is not recommended because doing so can lead to problematic behavior that disables them and could prevent the user from getting to their device. This is especially likely to happen when Windows is upgraded.

It has been common practice when wrapping a credential provider supplied by the operating system to develop a credential provider filter to disable the credential provider, such that only the wrapper is visible and active and the wrapper provider is not. However, using a filter to prevent a user from using a credential provider supplied by the operating system is not supported. There are policy controls available that allow IT admins to filter these credential providers.

Filtering of credential providers should be an enterprise choice so there are policy controls available that allow IT admins to filter credential providers. Please reference the Exclude credential providers Group Policy found under Computer Configurations¥Administrative Templates¥System¥Logon for more information.

Appendix E – LogonUI image display dimensions

In Windows Vista/7, the image size of the V1 Credential Provider tile is 126x126.

In Windows 8/8.1, the image size of user tile is 200x200, V1 Credential Provider tile continues to be 126x126, but will be centered inside a 200x200 pixel bounding box, background is RGB(70,70,70). The image size of V2 Credential Provider tile under selected users is 43x43.

In Windows 10, the selected User/V1/PLAP credential provider has an image size of 192x192. The ones on the bottom left list is of 48x48. Note LogonUI uses circular image for user and square image for V1/PLAP according to the new design direction. The image size of V2 Credential Provider tile under a selected user is 48x48.

Please consider updating the image resolution to make sure it does not look blurry on LogonUI.

Questions

Please contact credprov@microsoft.com with any questions.