

Credential Group Data and Services 5.0

# Document Status

status: Request for Comment (valid values are < Request for Comment, Preliminary Review, Public Review, Architectural Review, Final Review, Published, Deprecated)

# Authors and Change Log

|  |  |  |
| --- | --- | --- |
| Version | Date | Changes |
| 0.0.01 |  | * Initial Draft |
| 3.0 | **11/29/2013** | * Switch to use X-HTTP-METHOD-OVERRIDE standard rather than subMethod non-Standard method for overriding request types. * Create a credentialGroupMessage or a validateRegisteredDeviceMessage wrapper for every message to increase ability for infrastructure to serialize the data * Versioning and format change with release CUFX 3.0 |
| 3.0 | **12/13/2013** | * Update examples X-API-Version to >=3.0.0 |
|  | **12/19/2013** | * Enabled credentials to be verified * Enabled credentials to be related to each other * Renamed registered device specs to be more consistent and enable CRUD functionality * Completed rest of use cases examples * Updated definitions and overview |
| 3.1 | **07/17/2015** | * Updated to release 3.1 |
| 3.2 | **05/10/2016** | * Updated to release 3.2 |
| 3.3 | **02/15/2017** | * Updated to release 3.3 |
| 4.0 | **02/19/2018** | * Updated to release 4.0, Date Range Global Update, Microsoft Global bug fix |
| 4.1 | **12/10/2018** | * Updated to release 4.1, * CredentialGroup, ISOCountryCodeType, ISOCountryCodeType-v2006 - Replace all NMToken types with string * CredentialGroupFilter - Added Reference to User. Added userIdList, and usertypeList. |
| 4.2 | **03/05/2019** | * Updated to release 4.2, \*\*\* Release 4.2 is a breaking fix release. \*\*\* Errors found in App, ArtifactFilter, and BillFilter required a breaking fix to align with the standard and prevent additional implementation difficulties going forward. |
| 4.3 | **10/07/2019** | * Updated to release 4.3, renamed file removing version as proper version control is being used in Github. |
| 4.4 | **10/20/2020** | * Updated to release 4.4, CredentialGroup - Updated UserId to reference Common dataType. * CredentialGroupFilter - Added accountIdentificationList. |
| 4.5 | **04/02/2012** | * Updated to release 4.5, added support for Verifiable credentials, DID Validation and connection protocol for credential types. |
| 5.0 | **06/07/2021** | * Updated to release 5.0, CredentialGroupFilter - Update PartyIdList to reference Common.xsd. |

# Overview of Specification

The Credential Group Data Model and Services define how a credential group is created, read, updated and deleted on the core. A credential group is a combination of fields that allow a user access to a system and subsequently to a list of relationships and accounts.

# Any known Errors in the document

|  |  |
| --- | --- |
| **Error Description** | Status of Error |
|  |  |
|  |  |
|  |  |

# Table of Contents

[Document Status 1](#_Toc73693053)

[Authors and Change Log 1](#_Toc73693054)

[Overview of Specification 2](#_Toc73693055)

[Any known Errors in the document 2](#_Toc73693056)

[Table of Contents 2](#_Toc73693057)

[Document Conventions 3](#_Toc73693058)

[CUFX API and Documentation Support 3](#_Toc73693059)

[Release 4.0 Global Update Notes 4](#_Toc73693060)

[Release 4.4 Global Update Notes 4](#_Toc73693061)

[Release 5.0 Global Update Notes 5](#_Toc73693062)

[Definitions related to the specification 5](#_Toc73693063)

[Credential Group Services – CRUD Operations 6](#_Toc73693064)

[Overview 6](#_Toc73693065)

[Data Elements 6](#_Toc73693066)

[Filters used when accessing the Credential Group data 6](#_Toc73693067)

[CredentialGroup Data attributes 6](#_Toc73693068)

[Credential Group Resource based create, read, update, delete services 6](#_Toc73693069)

[REST-JSON CREATE Credential Group example 7](#_Toc73693070)

[REST-JSON READ Credential Group example 8](#_Toc73693071)

[REST-JSON UPDATE Credential Group example 9](#_Toc73693072)

[REST-JSON DELETE Credential Group example 10](#_Toc73693073)

[Credential Group Services – LMFA Services 11](#_Toc73693074)

[Overview 11](#_Toc73693075)

[High level use cases 12](#_Toc73693076)

[Use Case 1: Get Security Questions / Get Security Question/ Answer Security Question(s) 12](#_Toc73693077)

[Use Case 2: Get Confidence Picture/phrase 12](#_Toc73693078)

[Use Case 3: Select Verification Type options based on User profile and SUBMIT Choice – Activate Verification process 13](#_Toc73693079)

[Use Case 4: ENTER Security ACCESS Code 13](#_Toc73693080)

[Use caSE 5: Confirm Registry of browser/device 14](#_Toc73693081)

[Use Case 6: Register browser/device 14](#_Toc73693082)

[Use Case 7: Did Validation 14](#_Toc73693083)

[Use Case 8: Secure Connection 14](#_Toc73693084)

[Use Case 9: Verfied Credential 15](#_Toc73693085)

[Service Message: Use Case 2.1 GET Confidence Picture /Phrase 15](#_Toc73693086)

[Service Message: Use Case 5.1 Confirm Registered Browser/Device 15](#_Toc73693087)

[REST-JSON Example 16](#_Toc73693088)

[Service Message: Use Case 1.1, 1.2, 1.3 GET Defaulted Security Question(s) or a list of Security questions for user to choose from 16](#_Toc73693089)

[Service Message: Use Case 1.1 – 1.4 POST Security Question(s) Answer 17](#_Toc73693090)

[REST-JSON Example 17](#_Toc73693091)

[Service Message: Use Case 3.1 Request list of verification methods 19](#_Toc73693092)

[Service Message: Use Case 4.1-4.5 Verify Security Access Code 19](#_Toc73693093)

[Service Message: Use Case 4.5 Client Verify Security Access Code successfully done through Phone app push 19](#_Toc73693094)

[Service Message: Use Case 6.1 Register Browser/Device 19](#_Toc73693095)

[REST-JSON Example 20](#_Toc73693096)

[Service Message: Use Case 7.1 Did validation 21](#_Toc73693097)

[REST-JSON Example 21](#_Toc73693098)

[Service Message: Use Case 8.1 Secure Connection 23](#_Toc73693099)

[REST-JSON Example 23](#_Toc73693100)

[Service Message: Use Case 9.1 Verified Credential 26](#_Toc73693101)

[REST-JSON Example 26](#_Toc73693102)

[General Error handling For All Services 28](#_Toc73693103)

[Bibliography 28](#_Toc73693104)

# Document Conventions

List any document conventions such as what bold and italics mean and how the document is intended to be read.

“Within this specification, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in W3 Working Group (W3C)]. However, for readability, these words do not appear in all uppercase letters in this specification.

At times, this specification recommends good practice for authors and user agents. These recommendations are not normative and conformance with this specification does not depend on their realization. These recommendations contain the expression "We recommend ...", "This specification recommends ...", or some similar wording.”

All formatting in this document utilizes Word Styles.

All Citations must utilize Word Citations so that it automatically shows at the end of the document.

All updates after the initial creation must be performed using Tracking Changes turned on and accepted by the Architecture committee.

# CUFX API and Documentation Support

CUFX is published to SwaggerHub at <https://app.swaggerhub.com/apis/dlacroix01/CUFX> . The latest default version will load automatically.

The purpose of this publication is to demonstrate the full range of CUFX messaging. Provide for complete documentation of the entire message structure and example usage.

Swaggerhub as a normal support feature also translates into several dozen of the most common and in demand client / server formats. This provides a technology specific version of the CUFX standard for essentially all platforms.

The CUFX Specification documents have been written to provide for limited examples of case usage but do not reflect the entirety of elements present in any given message. Please see the latest version of CUFX in Swaggerhub for the complete message and for superior documentation support.

# Release 4.0 Global Update Notes

CUFX Release 4.0 introduces a number modifications that significantly improves the standard and is not backward compatible with prior versions.

Messaging paradigm shift. Prior to CUFX 4.0 a Message Object would be sent and would expect the Object List to be returned or the error message. The response had to be interrogated to determine what was received. With CUFX 4.0, the Object Message that is sent is also expected to be the Object that is returned. Significant improvements have been made to the Message Context to fully support Success, Informational, Warnings and Error responses. End Points may continue to use the prior methods, but use of the Error.xsd is depreciated; all functionality has transitioned into MessageContext.xsd.

Date Range Filtering. A global update was applied across the standard to remove the pairs of date filter elements for any given range and replaced with a single Common.xsd definition DateRange complex type. This makes date range filtering completely uniform across the standard and associates the startDateTime and endDateTime together as an object set.

As example: elements transactionStartDateTime and transactionEndDateTime were replaced in the AccountFilter.xsd with transactionDateRange.

Microsoft Serialization Bug. We discovered the root cause of a serialization error impacting CUFX. A known Microsoft Serialization error from 2006 is present for single element complex types. It causes a naming error of the serialized constructs. If both endpoints are using a Microsoft compilation the error is consistent and does not present itself, the names are both wrong but pass data successfully. When one end point is not using a Microsoft compilation, the field names are in variance and fails. If both end points are using non-Microsoft compilation the serialization would be correct and match.

CUFX 4.0 has applied a global update across all list types throughout the standard. The CUFX list construct was consistently a single element complex type. For all occurrences we have applied an extension base of common:ListBase. ListBase provides pagination support and also resolves the Microsoft serialization error. No longer being a single element complex type, Microsoft compilation now generates the correct names. This will necessitate prior (Microsoft) implementations to remap to the correct serialized names.

# Release 4.4 Global Update Notes

CUFX Release 4.4 introduces a significant enhancement for complex Account identification and filter navigation. The foundational architectural design premise for account navigation is that the CUFX AccountId would be a unique value unto itself within a given institution, or that a composite unique key would be passed. With the direct support of several core system providers it was established that that later case is predominate. The AccountId is generally not a unique value unless in combination with several other values such as AccountType and AccountSubType. Hence passing a unique AccountId meant that the organization had to overload the element value making filtering implementation specific and forcing the endpoints to map overlay the accountId to unpack the value.

CUFX now fully recognizes this architectural paradigm while continuing to support the original architecture.

A global update was applied to provide the elements accountType and accountSubType in all objects that contained accountId.

A new filter list – AccountIdentificationList has been added to all account related filters so that AccountId, AccountType, and AccountSubType can be structured properly for discreet filtering of complex account keys, support filtering by the sub keys and also support inbound and outbound account filtering using the accountToFromIndicator.

By expanding the architectural paradigm to support a non-unique AccountId CUFX is now positioned to better support core adoption of the standard.

# Release 5.0 Global Update Notes

CUFX Release 5.0 Moved PartyId and PartyIdList into the Common.xsd. Namespaces references to Party were removed if there were no other references to Party in the specification. PartyId and PartyIdList type references were updated.

# Definitions related to the specification

**Credential Group**

Credential group is the information known only by the party used to gain access to their data. Credential Groups are related to the party. Access to a given account related to party is based on their role on the account.

**Registered Device**

A registered Device is a device a user uses such as a phone, tablet, PC, etc. to access their account information. The device can be related to a list of accounts, relationships or parties.

# Credential Group Services – CRUD Operations

## Overview

|  |  |
| --- | --- |
| Definition | Collection of services to manage a credential group or list of credential groups |
| Overview of Capabilities | Create, read, update and delete a credential group. |
| Dependencies | Security Services, messageContext, party |
| Sample CUFX REST LINK | https://api.dataprovider.com/credentialgroupmessage |
| CUFX SOAP LINK |  |
| CUFX WaDL LINK |  |

# Data Elements

## Filters used when accessing the Credential Group data

Refer to Security Services documentation to understand what may be contained in the header and processed by security procedures. When accessing the data include **MessageContext.xsd** so that the service can determine the scope of the request. Refer to recent CUFX messageContext Data and CUFX Security Services for use of MessageContext.xsd. Include any filter variables related to the request. See **CredentialGroupFilter.xsd.**

## CredentialGroup Data attributes

All CUFX fields related to a credential group are defined in **CredentialGroup.xsd.**

Note: Fields not listed in the calling specification are not to be returned to the calling specification. That is, If the field transaction type is not listed in the calling specification, then do not return the data field to alleviate issues with unexpected information and a bloat of information being returned to light weight applications.

## Credential Group Resource based create, read, update, delete services

|  |  |
| --- | --- |
| INPUTS | cufx:credentialGroupMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx:credentialGroupFilter (for read, update) * cufx:credentialGroupList (for create, update, delete) |
| Outputs | cufx:credentialGroupMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx:credentialGroupList |
| Return Values | cufx:credentialGroupMessage (which includes)   * cufx:MessageContext   + statusList |
| Side Effects | Creation, update or deletion of credentialGroup; read has no side effects. |
| Dependencies | Security Services for authentication and security. |
| Fields used | Message Headers : See security services  messageContext: See MessageContext.xsd  Filters: See CredentailGroupFilter.xsd  Attributes: credentialGroup: See CredentialGroup.xsd |

### REST-JSON CREATE Credential Group example

This example creates a credential group. The party must have the fiUserId added to it to connect it to the credentialGroup.

Note: Not all fields are listed for simplicity of an example to create a credential group.

**Required**: messageContext, at least one credential group within CredentialGroupList.

**REQUEST:**

Headers:

**<security related header parameters... see Security Services>**

Accept: application/json

Accept-Charset: utf-8

Accept-Language: en-us (IANA – language codes)(W3C, HTTP Protocols)

Content-type: application/json; charset=utf-8

X-API-Version: >=5.0.0

**POST h**ttps://api.datasource.com/credentialgroupmessage

{

“credentialGroupMessage”: {

“messageContext”: { <see MessageContext.xsd>

},

“credentialGroupList”: [

{“credentialList”: [

{“Type”:”Username”,

“Value”:”tomthumb”

},

{“Type”:”Password”,

“Value”:”1513KADFIkf&82j5”

“EncryptedFlag”:”true”

}

],

“deliveryChannelList”:[

{“deliveryChannel”:”OnlineBanking”},

{“deliveryChannel”:”Mobile”}

],

},

{“credentialList”:[

{“Type”:”SecurityQuestion”,

“Value”:”What is your favorite Car Model?”

},

{“Type”:”SecurityAnswer”,

“Value”:”1513KADFIkf&82j5”

“EncryptedFlag”:”true”

}

],

“deliveryChannelList”:[

{“deliveryChannel”:”OnlineBanking”},

]

}

]

}

}

**RESPONSE:**

Headers:

Status Code: 200 Ok

Content-type: application/json; charset=utf-8

Content-Language: en-us

Payload:

{

“credentialGroupMessage”: {

“messageContext”: { <see MessageContext.xsd>

},

“credentialGroupList”: [

{“fiUserId”:”84541abs38375443”,

“credentialList”: [

{“Type”:”Username”,

“Value”:”tomthumb”

},

{“Type”:”Password”,

“Value”:”1513KADFIkf&82j5”

“EncryptedFlag”:”true”

}

],

“deliveryChannelList”:[

{“deliveryChannel”:”OnlineBanking”},

{“deliveryChannel”:”Mobile”}

],

},

{“fiUserId”:”183563pijef”,

“credentialList”:[

{“Type”:”SecurityQuestion”,

“Value”:”What is your favorite Car Model?”

},

{“Type”:”SecurityAnswer”,

“Value”:”1513KADFIkf&82j5”

“EncryptedFlag”:”true”

}

],

“deliveryChannelList”:[

{“deliveryChannel”:”OnlineBanking”},

]

}

]

}

}

### REST-JSON READ Credential Group example

Note: Not all fields are listed for simplicity of an example to read a credential group.

**Required**: messageContext, at least one valid filter in credentialGroupFilter.

**REQUEST:**

Headers:

**<security related header parameters... see Security Services>**

Accept: application/json

Accept-Charset: utf-8

Accept-Language: en-us (IANA – language codes)(W3C, HTTP Protocols)

Content-type: application/json; charset=utf-8

**X-HTTP-Method-Override: GET**

X-API-Version: >=5.0.0

**POST h**ttps://api.datasource.com/credentialgroupmessage

{

“credentialGroupMessage”: {

“messageContext”: { <see MessageContext.xsd>

},

“credentialGroupFilter”:{

“fiUserIdList”:[

”84541abs38375443”,

“183563pijef”

]

}

}

}

**RESPONSE:**

Headers:

Status Code: 200 Ok

Content-type: application/json; charset=utf-8

Content-Language: en-us

Payload:

{

“credentialGroupMessage”: {

“messageContext”: { <see MessageContext.xsd>

},

“credentialGroupList”: [

{“fiUserId”:”84541abs38375443”,

“credentialList”: [

{“Type”:”Username”,

“Value”:”tomthumb”

},

{“Type”:”Password”,

“Value”:”1513KADFIkf&82j5”

“EncryptedFlag”:”true”

}

],

“deliveryChannelList”:[

{“deliveryChannel”:”OnlineBanking”},

{“deliveryChannel”:”Mobile”}

],

},

{“fiUserId”:”183563pijef”,

“credentialList”:[

{“Type”:”SecurityQuestion”,

“Value”:”What is your favorite Car Model?”

},

{“Type”:”SecurityAnswer”,

“Value”:”1513KADFIkf&82j5”

“EncryptedFlag”:”true”

}

],

“deliveryChannelList”:[

{“deliveryChannel”:”OnlineBanking”},

]

}

]

}

}

### REST-JSON UPDATE Credential Group example

Note: Not all fields are listed for simplicity of an example to update a credential group.

**Required**: messageContext, at least one valid filter in credentialGroupFilter, the specific id of the fiUserId to be updated in the message.

**REQUEST:**

Headers:

**<security related header parameters... see Security Services>**

Accept: application/json

Accept-Charset: utf-8

Accept-Language: en-us (IANA – language codes)(W3C, HTTP Protocols)

Content-type: application/json; charset=utf-8

X-API-Version: >=5.0.0

**PUT h**ttps://api.datasource.com/credentialgroupmessage

{

“credentialGroupMessage”: {

“messageContext”: { <see MessageContext.xsd>

},

“credentialGroupFilter”:{

“fiUserIdList”:[

”84541abs38375443”

]

},

“credentialGroupList”: [

{“fiUserId”:”84541abs38375443”,

“credentialList”: [

{“Type”:”Password”,

“Value”:”18ad#$%%#5877981jlk1233==”

“EncryptedFlag”:”true”

}

]

}

]

}

}

**RESPONSE:**

Headers:

Status Code: 200 Ok

Content-type: application/json; charset=utf-8

Content-Language: en-us

Payload:

{

“credentialGroupMessage”: {

“messageContext”: { <see MessageContext.xsd>

},

“credentialGroupList”: [

{“fiUserId”:”84541abs38375443”,

“credentialList”: [

{“Type”:”Username”,

“Value”:”tomthumb”

},

{“Type”:”Password”,

“Value”:”18ad#$%%#5877981jlk1233==”

“EncryptedFlag”:”true”

}

],

“deliveryChannelList”:[

{“deliveryChannel”:”OnlineBanking”},

{“deliveryChannel”:”Mobile”}

]

}

]

}

}

### REST-JSON DELETE Credential Group example

Note: This example deletes a credential group.

**Required**: messageContext, at least one valid filter in credentialGroupFilter

**REQUEST:**

Headers:

**<security related header parameters... see Security Services>**

Accept: application/json

Accept-Charset: utf-8

Accept-Language: en-us (IANA – language codes)(W3C, HTTP Protocols)

Content-type: application/json; charset=utf-8

**X-HTTP-Method-Override: DELETE**

X-API-Version: >=5.0.0

**PUT h**ttps://api.datasource.com/credentialgroupmessage

{

“credentialGroupMessage”: {

“messageContext”: { <see MessageContext.xsd>

},

“credentialGroupFilter”:{

“fiUserIdList”:[

”84541abs38375443”

]

}

}

}

**RESPONSE:**

Headers:

Status Code: 200 Ok

Content-type: application/json; charset=utf-8

Content-Language: en-us

Payload:

{

"credentialGroupMessage": {

"@xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",

"messageContext": {

"cufxVersion": "4.0.0",

"requestId": "requestId1",

"vendorId": "vendorId1",

"appId": "appId1",

"fiId": "84541abs38375443",

"dataSourceId": "dataSourceId1",

"environment": "Development",

"returnDataFilter": "All",

"includeBlankFields": "true",

"includeZeroNumerics": "true",

"user": {

"userId": "userId1",

"processorSessionId": "processorSessionId1",

"userType": "EmployeeId"

},

"statusList": {

"status": {

"statusType": "Success"

}

}

}

}

}

# Credential Group Services – LMFA Services

## Overview

|  |  |
| --- | --- |
| Definition | Collection of services related to Layered Multifactor |
| Overview of Capabilities | Store and retrieve information about devices so that an additional layer of security is provided based on what the user has and what they know. |
| Dependencies | CredentialGroup |
| Sample CUFX REST LINK | <https://api.dataprovider.com/credentialgroup>message  <https://api.dataprovider.com/registereddevice>message |
| CUFX SOAP LINK |  |
| CUFX WaDL LINK |  |

High level use cases

Use Case 1: Get Security Questions / Get Security Question/ Answer Security Question(s)

Use Case 1.1 Defaulted single security question

1. User interface/application subsystem calls service to get a valid security question.
2. Security question is displayed for user to enter answer.
3. Answer is entered and submitted to the service method for verification.

Use Case 1.2 Defaulted multiple security questions

1. User interface/application subsystem calls service to get a valid list of security questions.
2. Security questions are displayed for user to enter answers.
3. User enters answers for each question and submits to server for verification.

Use Case 1.3 User choose single/multiple security questions

1. User interface/application subsystem calls service to get list of security question descriptions associated with profile.
2. User interface displays security question list for user and user to choose N out of total questions to answer.
3. User chooses required number of security questions. Example 3 out of 5 questions associated with profile.
4. User enters answers for each question and submits to server for verification.

Use Case 2: Get Confidence Picture/phrase

Use Case 2.1

1. User interface/application subsystem calls service to get confidence picture (id or url) and confidence phrase. These are used to prevent phishing attempts (Referred to as AntiPhishing in the schema).
2. User interface displays confidence picture associated with given account, person, etc. and confidence phrase

Use Case 3: Select Verification Type options based on User profile and SUBMIT Choice – Activate Verification process

Use case 3.1 User choice verification method

1. User interface/application subsystem calls service to get list of possible verification methods. This is often done after a Use Case 5 response of unregistered device.
2. User interface displays possible verification choices for user.
3. User chooses verification method and submits to the server to start verification method.
4. User Interface shows the user that the verification has been sent.

Use case 3.2 Defaulted method

1. User interface/application subsystem calls service to launch default verification method returning description of default method.
2. User Interface shows the user that the verification has been sent.

Use Case 4: ENTER Security ACCESS Code

Use Case 4.1 Email

1. User receives email with temporary security access code.
2. User clicks link in email or goes to previous UI screen and enters the temporary security access code.
3. UI submits code to the server for validation.

Use Case 4.2 SMS

1. User receives SMS message with temporary security access code.
2. User reads SMS message and enters the temporary security access code.
3. UI submits code to the server for validation.

Use Case 4.3 Voice

1. User receives Voice phone call message with temporary security access code.
2. User enters the temporary security access code.
3. UI submits code to the server for validation.

Use Case 4.4 Smart Phone Token

1. User is told to see Smart Phone Token application for access code.
2. User enters the security access code in UI.
3. UI submits code to the server for validation

Use Case 4.5 Smart Phone Push

1. User is told to see Smart Phone push.
2. User presses Smart Phone push to validate.
3. Smart phone send response to server to enable transaction (on computer).
4. User’s browser screen moves forward (by click or auto) and verifies phone push is in and user moves forward.

Use caSE 5: Confirm Registry of browser/device

Use Case 5.1

1. User interface/application subsystem calls service to see if current client is registered.
2. Service returns information and workflow continues accordingly. If approved no verification process. If not approved, verification process is started.

Use Case 6: Register browser/device

Use Case 6. 1

1. User has gone through access verification process and presented with question to add current browser/device to registered browser/device list.
2. User choose to approve (or not) and UI calls server to call service to record approval.

Use Case 7: Did Validation

Use Case 7. 1

1. User or third party is presenting a decentralized identifier for validation.
2. Receiving end point conducts the validation.
3. Service returns the verified credential and workflow continues accordingly.

Use Case 8: Secure Connection

Use Case 8. 1

1. User or third party is presenting a secure connection protocol message.
2. Service responds with the corresponding secure connection protocol response message and workflow continues accordingly.

Use Case 9: Verfied Credential

Use Case 9. 1

1. User or third party has conducted a decentralized Id validation and is presenting a verified credential.
2. Service evaluates the credential and workflow continues accordingly.

### Service Message: Use Case 2.1 GET Confidence Picture /Phrase

This message is sent by the client to request the Confidence Picture / Phrase information. This feature is currently handled by the Credentials group READ functionality described earlier.

The available filters CredentialTypes for “anti-phishing” are

* AntiphishingPhrase
* AntiphishingImageUrl
* AntiphishingImageArtifactId (This is a pointer to the artifact repository)

The application would have to know all the available phrases or images but credentialGroup would have the correct phrase or image for the account being accessed.

### Service Message: Use Case 5.1 Confirm Registered Browser/Device

This message is used by the client to verify that the user’s browser/device is a registered entity.

|  |  |
| --- | --- |
| INPUTS | cufx: registeredDeviceMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: registeredDeviceFilter (for read, update) * cufx:cregisteredDeviceList (for create, update, delete) |
| Outputs | cufx: registeredDeviceMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: registeredDeviceList |
| Return Values | cufx: registeredDeviceMessage (which includes)   * cufx:MessageContext   + statusList |
| Side Effects | None |
| Dependencies |  |
| Fields Used | See registeredDeviceFilter.xsd and registeredDevice.xsd |

REST-JSON Example

**Required**: messageContext, validateRegisteredDeviceRequest.

**REQUEST:**

Headers:

**<security related header parameters... see Security Services>**

Accept: application/json

Accept-Charset: utf-8

Accept-Language: en-us (IANA – language codes)(W3C, HTTP Protocols)

Content-type: application/json; charset=utf-8

**X-HTTP-Method-Override: GET**

X-API-Version: >=5.0.0

**POST h**ttps://api.datasource.com/registereddevicemessage

{

“r**egistered**DeviceMessage”: {

“messageContext”: { <see MessageContext.xsd>

},

“registeredDeviceFilter”: {

“registeredDeviceIdList”:[

“12755225523455qaat3255338656

]

}

}

}

**RESPONSE:**

Headers:

Status Code: 200 Ok

Content-type: application/json; charset=utf-8

Content-Language: en-us

Payload:

{

“registeredDeviceList”: {

“registeredDevice”: {

“accountIdList:{“135341234”,”35534”},

“deviceID”:”device/ip identification string”,

“encryptedFlag”:”true”,

“isRegisteredFlag”: “true”

}

}

}

### Service Message: Use Case 1.1, 1.2, 1.3 GET Defaulted Security Question(s) or a list of Security questions for user to choose from

This message is used to request from the service the default required service security question, default required service security questions, or a list of security questions associated with a credential group for a user to choose X number of the total questions to answer.

This feature is currently handled by the credentials group READ functionality described earlier. A filter would be used to query the service for a credential group list of the type Security Questions. The filter could also filter by Deliver Channel.

The available filters CredentialTypes for “Security Questions” are

* SecurityQuestion
* SecurityAnswer

The application would have to know all the available phrases or images but credentialGroup would have the correct phrase or image for the account being accessed.

It is up to the application to determine if there is a predetermined set of questions that can be answered or is it up to the user to set their own questions.

Depending upon the agreement with the data provider, the application may query for the security questions and then compare if the security answers match the security questions. Encryption can be applied to the security answers if needed.

If the security answer is not returned by the data provider, the application may need to execute a CredentialGroupFilter with the verifyCredentialGroupList set. If this is set, then the back end system can verify the credential values are correct. If so, the verifyCredentialGroupToken field will be set with a value.

### Service Message: Use Case 1.1 – 1.4 POST Security Question(s) Answer

This message is used to post the response of the security question(s) for validation.

|  |  |
| --- | --- |
| INPUTS | cufx: credentialGroupMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: credentialGroupFilter (for read, update) * cufx: credentialGroupList (for create, update, delete) |
| Outputs | cufx: credentialGroupMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: credentialGroupList |
| Return Values | cufx:credentialGroupMessage (which includes)   * cufx:MessageContext   + statusList |
| Side Effects | Typically read has no side effect. However, this process might save a token in the back end system representing the verified session. |
| Dependencies |  |
| Fields Used | See CredentialsGroupFilter.xsd and CredentialGroup.xsd |

REST-JSON Example

This example validate that the user gave the correct answer for the security questions. A user’s answer in the request is mapped to a credential list identifier. It supports multiple security questions being answered.

Note: Not all fields are listed for simplicity of an example to create a credential group.

**Required**: messageContext, at least one entry in verifyCredentialGroupList.

**REQUEST:**

Headers:

**<security related header parameters... see Security Services>**

Accept: application/json

Accept-Charset: utf-8

Accept-Language: en-us (IANA – language codes)(W3C, HTTP Protocols)

Content-type: application/json; charset=utf-8

**X-HTTP-Method-Override: GET**

X-API-Version: >=5.0.0

**POST h**ttps://api.datasource.com/credentialgroupmessage

{

“credentialGroupMessage”: {

“messageContext”: { <see MessageContext.xsd>

},

“credentialGroupFilter”: [

{“verifyCredentialGroupList”:[

{“credentialList”:[

{“credentialId:”1353434151234”,

“relatedCredentialId:”46452452345”,

“Type”:”SecurityQuestion”,

“Value”:”What is your favorite Car Model?”

},

{“credentialId”:”46452452345”,

“Type”:”SecurityAnswer”,

“Value”:”1513KADFIkf&82j5”

“EncryptedFlag”:”true”

}

],

“deliveryChannelList”:[

{“deliveryChannel”:”OnlineBanking”},

]

}

]

}

}

**RESPONSE:**

Headers:

Status Code: 200 Ok

Content-type: application/json; charset=utf-8

Content-Language: en-us

Payload:

{

{“credentialGroupList”:[

{“verifiedCredentialGroupToken”:1345aash1535432-13512h=-8351234-35h34”

“credentialList”:[

{“credentialId:”1353434151234”,

“relatedCredentialId:”46452452345”,

“Type”:”SecurityQuestion”,

“Value”:”What is your favorite Car Model?”

},

{“credentialId”:”46452452345”,

“Type”:”SecurityAnswer”,

“Value”:”1513KADFIkf&82j5”

“EncryptedFlag”:”true”

}

],

“deliveryChannelList”:[

{“deliveryChannel”:”OnlineBanking”},

]

}

]

}

}

### Service Message: Use Case 3.1 Request list of verification methods

This message is used to request from the service the credential list verification methods that are available in the application. This list is used to give the user a choice of which verification method they would like to use that is associated with their profile.

This feature is currently handled by the credentials group READ functionality described earlier. A filter would be used to query the service for a credential group list of credential list types. The filter also would filter by Deliver Channel.

### Service Message: Use Case 4.1-4.5 Verify Security Access Code

This message is used by the client to verify that the security access code that was given by the verification process. Validates the code and signals that the verification was successful or not.

This feature is similar to the earlier verification of the security question/answer. However, the verify would use the CredentialType = “AuthorizationCode”. The credential would most likely have the temporary field set to True. The response to the verification would be a verifiedCredentialGroupToken that could be used in subsequent steps to move the process forward.

### Service Message: Use Case 4.5 Client Verify Security Access Code successfully done through Phone app push

This message is used by the client to verify that the phone push was done and verification is finished; client can move forward in registration process.

The verfieidCredentialGroupToken would be used in the messageContext of future messages to provide authorization to any future CUFX function. The security services would validate that token based on agreed upon security methods.

### Service Message: Use Case 6.1 Register Browser/Device

This message is used by the client to register the user’s browser/device for future inquiries.

|  |  |
| --- | --- |
| INPUTS | cufx: registeredDeviceMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: registeredDeviceFilter (for read, update) * cufx:cregisteredDeviceList (for create, update, delete) |
| Outputs | cufx: registeredDeviceMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: registeredDeviceList |
| Return Values | cufx:credentialGroupMessage (which includes)   * cufx:MessageContext   + statusList |
| Side Effects | None |
| Dependencies |  |
| Fields Used | See fields in registeredDevice.xsd |

REST-JSON Example

**Required**: messageContext, validateRegisteredDeviceRequest.

**REQUEST:**

Headers:

**<security related header parameters... see Security Services>**

Accept: application/json

Accept-Charset: utf-8

Accept-Language: en-us (IANA – language codes)(W3C, HTTP Protocols)

Content-type: application/json; charset=utf-8

X-API-Version: >=5.0.0

**POST h**ttps://api.datasource.com/registereddevicemessage

{

“r**egistered**DeviceMessage”: {

“messageContext”: { <see MessageContext.xsd>

},

“registeredDeviceList”: [

{“accountIdList:{“135341234”,”35534”},

“deviceID”:”device/ip identification string”,

“encryptedFlag”:”true”,

“isRegisteredFlag”: “true”

}

]

}

}

**RESPONSE:**

Headers:

Status Code: 200 Ok

Content-type: application/json; charset=utf-8

Content-Language: en-us

Payload:

{

“registeredDeviceMessage”: {

“messageContext”: { <see MessageContext.xsd>

},

“registeredDeviceList”: [

{“accountIdList:{“135341234”,”35534”},

“deviceID”:”device/ip identification string”,

“encryptedFlag”:”true”,

“isRegisteredFlag”: “true”

}

]

}

}

### Service Message: Use Case 7.1 Did validation

This message is used to post decentralized identifer for validation.

|  |  |
| --- | --- |
| INPUTS | cufx: credentialGroupMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: credentialGroupFilter (for read, update) * cufx: credentialGroupList (for create, update, delete) |
| Outputs | cufx: credentialGroupMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: credentialGroupList |
| Return Values | cufx:credentialGroupMessage (which includes)   * cufx:MessageContext   + statusList |
| Side Effects | Typically read has no side effect. However, this process might save a token in the back-end system representing the verified session. |
| Dependencies |  |
| Fields Used | See CredentialsGroupFilter.xsd and CredentialGroup.xsd |

REST-JSON Example

This example the user or third party is presenting a decentralized Identifier for validation.

Note: Not all fields are listed for simplicity of an example to create a credential group.

**Required**: messageContext, at least one entry in credentialGroupList.

**REQUEST:**

Headers:

**<security related header parameters... see Security Services>**

Accept: application/json

Accept-Charset: utf-8

Accept-Language: en-us (IANA – language codes)(W3C, HTTP Protocols)

Content-type: application/json; charset=utf-8

X-API-Version: >=5.0.0

**POST** https://api.datasource.com/credentialgroupmessage

{

"credentialGroupMessage": {

"messageContext": {

"cufxVersion": "4.5",

"requestId": "123",

"sessionId": "string",

"userList": {

"user": [

{

"userId": "123453566",

"processorSessionId": "string",

"userIdType": "EmployeeId",

}

]

}

},

"credentialGroupList": {

"credentialGroup": [

{

"userId": "123453566",

"userIdType": "EmployeeId",

"credentialList": {

"credential": [

{

"credentialId": "1",

"credentialType": "DidValidation",

"encrypted": false,

"value": "did:example:123456789abcdefghi#keys-1"",

},

"didValidation": {

"id": "did:example:123456789abcdefghi#keys-2",

"controller": "did:example:123456789abcdefghi",

"type": "Ed25519VerificationKey2018",

"publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"

}

}

]

}

}

]

}

}

}

**RESPONSE:**

Headers:

Status Code: 200 Ok

Content-type: application/json; charset=utf-8

Content-Language: en-us

Payload:

{

"credentialGroupMessage": {

"messageContext": {

"cufxVersion": "4.5",

"requestId": "123",

"user": [

{

"userId": "123453566",

"processorSessionId": "string",

"userIdType": "EmployeeId

}

]

},

"statusList": {

"status": [

{

"statusType": "Success",

}

]

}

},

"credentialGroupList": {

"credentialGroup": [

{

"userId": "123453566",

"userIdType": "EmployeeId",

"credentialList": {

"credential": [

{

"credentialId": "1",

"relatedCredentialId": "string",

"credentialType": "VerifiedCredential",

"verifiedCredential": {

"attributeList": {

"attribute": [

{

"name": "Institution",

"value": "InstitutionName"

}

],

"attribute": [

{

"name": "EmployeeID",

"value": "123453566"

}

]

}

}

}

]

}

"verifiedCredentialGroupToken": "string"

}

]

}

}

}

### Service Message: Use Case 8.1 Secure Connection

This message is used to pass a secure connection protocol message and response.

|  |  |
| --- | --- |
| INPUTS | cufx: credentialGroupMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: credentialGroupFilter (for read, update) * cufx: credentialGroupList (for create, update, delete) |
| Outputs | cufx: credentialGroupMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: credentialGroupList |
| Return Values | cufx:credentialGroupMessage (which includes)   * cufx:MessageContext   + statusList |
| Side Effects | Typically read has no side effect. However, this process might save a token in the back-end system representing the verified session. |
| Dependencies |  |
| Fields Used | See CredentialsGroupFilter.xsd and CredentialGroup.xsd |

REST-JSON Example

This example the user or third party is initiating a secure connection protocol and response.

**Required**: messageContext, at least one entry in credentialGroupList.

**REQUEST:**

Headers:

**<security related header parameters... see Security Services>**

Accept: application/json

Accept-Charset: utf-8

Accept-Language: en-us (IANA – language codes)(W3C, HTTP Protocols)

Content-type: application/json; charset=utf-8

X-API-Version: >=5.0.0

**POST** https://api.datasource.com/credentialgroupmessage

{

"credentialGroupMessage": {

"messageContext": {

"cufxVersion": "4.5",

"requestId": "123",

"sessionId": "string",

"userList": {

"user": [

{

"userId": "123453566",

"processorSessionId": "string",

"userIdType": "EmployeeId",

}

]

}

},

"credentialGroupList": {

"credentialGroup": [

{

"userId": "123453566",

"userIdType": "EmployeeId",

"credentialList": {

"credential": [

{

"credentialId": "1",

"credentialType": "ConnectionProtocol",

},

"connectionProtocol": {

"protocolName": "aries-rfcs",

"protocolFeatureName": "RFC 0113: Question Answer Protocol",

"protocolVersion": "0.9",

"protocolMessage": "{

"@type": "https://didcomm.org/questionanswer/1.0/question",

"@id": "518be002-de8e-456e-b3d5-8fe472477a86",

"question\_text": "Alice, are you on the phone with Bob from Faber Bank right now?",

"question\_detail": "This is optional fine-print giving context to the question and its various answers.",

"nonce": "<valid\_nonce>",

"signature\_required": true,

"valid\_responses" : [

{"text": "Yes, it's me"},

{"text": "No, that's not me!"}],

"~timing": {

"expires\_time": "2018-12-13T17:29:06+0000"

}

}"

}

}

]

}

}

]

}

}

}

**RESPONSE:**

Headers:

Status Code: 200 Ok

Content-type: application/json; charset=utf-8

Content-Language: en-us

Payload:

{

"credentialGroupMessage": {

"messageContext": {

"cufxVersion": "4.5",

"requestId": "123",

"user": [

{

"userId": "123453566",

"processorSessionId": "string",

"userIdType": "EmployeeId

}

]

},

"statusList": {

"status": [

{

"statusType": "Success",

}

]

}

},

"credentialGroupList": {

"credentialGroup": [

{

"userId": "123453566",

"userIdType": "EmployeeId",

"credentialList": {

"credential": [

{

"credentialId": "1",

"relatedCredentialId": "string",

"credentialType": " ConnectionProtocol",

"verifiedCredential": {

"connectionProtocol": {

"protocolName": "aries-rfcs",

"protocolFeatureName": "RFC 0113: Question Answer Protocol",

"protocolVersion": "0.9",

"protocolResponse": "{

"@type": "https://didcomm.org/questionanswer/1.0/answer",

"~thread": { "thid": "518be002-de8e-456e-b3d5-8fe472477a86", "seqnum": 0 },

"response": "Yes, it's me",

"response~sig": {

"@type": "https://didcomm.org/signature/1.0/ed25519Sha512\_single"

"signature": "<digital signature function output>",

"sig\_data": "<base64(HASH("Alice, are you on the phone with Bob?"+"Yes, it's me"+"<nonce>"))>",

"signers": ["<responder\_key>"],

}

"~timing": {

"out\_time": "2018-12-13T17:29:34+0000"

}

}"

}

}

]

}

"verifiedCredentialGroupToken": "string"

}

]

}

}

}

### Service Message: Use Case 9.1 Verified Credential

This message is used to post did for validation.

|  |  |
| --- | --- |
| INPUTS | cufx: credentialGroupMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: credentialGroupFilter (for read, update) * cufx: credentialGroupList (for create, update, delete) |
| Outputs | cufx: credentialGroupMessage (which includes)   * [cufx:messageContext](file:///\\files2\users\CMarjaniemi\Projects\CUFX\MessageContext.html) * cufx: credentialGroupList |
| Return Values | cufx:credentialGroupMessage (which includes)   * cufx:MessageContext   + statusList |
| Side Effects | Typically read has no side effect. However, this process might save a token in the back-end system representing the verified session. |
| Dependencies |  |
| Fields Used | See CredentialsGroupFilter.xsd and CredentialGroup.xsd |

REST-JSON Example

This example the user or third party is presenting a verified credential for acceptance.

Note: Not all fields are listed for simplicity of an example to create a credential group.

**Required**: messageContext, at least one entry in credentialGroupList.

**REQUEST:**

Headers:

**<security related header parameters... see Security Services>**

Accept: application/json

Accept-Charset: utf-8

Accept-Language: en-us (IANA – language codes)(W3C, HTTP Protocols)

Content-type: application/json; charset=utf-8

X-API-Version: >=5.0.0

**POST** https://api.datasource.com/credentialgroupmessage

{

"credentialGroupMessage": {

"messageContext": {

"cufxVersion": "4.5",

"requestId": "123",

"sessionId": "string",

"userList": {

"user": [

{

"userId": "123453566",

"processorSessionId": "string",

"userIdType": "EmployeeId"

}

]

}

},

"credentialGroupList": {

"credentialGroup": [

{

"userId": "123453566",

"userIdType": "EmployeeId",

"credentialList": {

"credential": [

{

"credentialId": "1",

"credentialType": "VerifiedCredential",

"verifiedCredential": {

"attributeList": {

"attribute": [

{

"name": "Institution",

"value": "InstitutionName"

}

],

"attribute": [

{

"name": "EmployeeID",

"value": "123453566"

}

]

}

}

}

]

}

}

]

}

}

}

**RESPONSE:**

Headers:

Status Code: 200 Ok

Content-type: application/json; charset=utf-8

Content-Language: en-us

Payload:

{

"credentialGroupMessage": {

"messageContext": {

"cufxVersion": "4.5",

"requestId": "123",

"user": [

{

"userId": "123453566",

"processorSessionId": "string",

"userIdType": "EmployeeId

}

]

},

"statusList": {

"status": [

{

"statusType": "Success",

}

]

}

},

"credentialGroupList": {

"credentialGroup": [

{

"userId": "123453566",

"userIdType": "EmployeeId",

"verifiedCredentialGroupToken": "string"

}

]

}

}

}

# General Error handling For All Services

Refer to latest CUFX documentation *Error Mapping*.

Bibliography

*E.164.* (n.d.). Retrieved 06 28, 2012, from International PUblic Telecommunications Number Plan: http://www.itu.int/rec/T-REC-E.164/en

*North American Number Plan Administration.* (n.d.). Retrieved 06 28, 2012, from North American Number Plan Administration: http://www.nanpa.com/

W3C. (n.d.). *Key words for use in RFCs to Indicate Requirement Levels [RFC2119].* Retrieved Sept. 8th, 2011, from W3C.