AERO SUPPORT CORPORATION

MTU Aero Engines

INTERFACE CONTROL DOCUMENT FOR

User Security System

DATE:

MDS RELEASED DOCUMENT

AUTH: [illegible]

LICENSED PROFESSIONAL ENGINEER
R. S. SCHROEDER
PROVINCE OF ONTARIO

DOCUMENT NUMBER	REVISION
ICD78024.2673	4

	NAME	POSITION	COMPANY	SIGNATURE	DATE
Prepared By:	Tigran Galoyan	Windows Software Developer	MDS	[illegible]	29-Jan-16
Reviewed By:	Robert Schroeder	Chief Software Engineer	MDS	Robert Schroeder	29-Jan-16
Reviewed By:	John Perrin	Manager, Software Engineering	MDS	[illegible]	29-Jan-16
Reviewed By:	Thomas Speer	Project Manager	MTU	[illegible]	29-Jan-16
Approved By:	Paul Suominen	Product Manager - proDAS	MDS	[illegible]	29-Jan-16

PUBLISHED BY: [illegible]

MDS Aero Support Corporation
1220 Old Innes Road, Suite 200, Ottawa, Ontario, Canada K1B 3V3
Phone (613) 744-7257 Fax (613) 744-8016

MTU Aero Engines GmbH
Dachauer Straße 665,
80995 München

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

1.1.1 The User Security System provides security to all proDAS applications. This specification defines the technical requirements for the interfaces offered by the User Security System.

Unlike the existing version of USS, for the new USS, at a high level, there will no longer be a DCOM server hosting security settings, rather a single security server hosting all the security settings, accessible via a client using WCF, with a WPF GUI client management application. All clients will be required to connect to the security server with the requirement for the support of disconnected security settings having been removed.

In addition to providing new features such as requesting resource details on demand, the main goals of the new implementation are to remove the legacy VB implementation of the existing version, to support the existing core concepts of the USS application and remove the use of client side raw passwords currently stored in configuration .INI files. To assist in the security requirements, a small core set of technologies will be used that are well understood so that shared code can be leveraged and security attack surface minimised.

One security related proposal that changes some of the existing calls is the idea of a security session for the USS application and an application session for other applications, both managed by the security server, so that management and user passwords are not sent with each request. Instead a call is made to get a security session with a user name and password, creating a short term (in the case of the USS application ~60 minute window, while other applications can request a session as long as 3 days) with ongoing calls requiring the client to use a valid session. It needs to be mentioned though, that the user has an option to extend the current USS application session by another 60 minutes. This flexibility allows the user to save locally made changes in case the session is expired within 60 minutes. The user just needs to verify its validity by logging in again. The session essentially ties a user name, application, permission to a client IP address bound by an expiry date time. Sessions are immutable in that their core details cannot be changed other than expired early with clients being able to expire their sessions when they are no longer needed. To support chaining applications together a session can be spawned for another application.

In addition to session verification, the current username and password verification will be supported, in that a user name and an encrypted password can be used to verify a user.

There are new login and change password controls implemented as a CCW [COM-Callable Wrapper] to mirror functionality that was in the preceding version. Following core technology and security guidelines the underlying UI will be implemented in WPF which has a password control that exposes a password as a .NET SecureString.

1.2 Scope

1.2.1 This document is intended for programmers who may use the functionality provided whether through the server side interface to send requests directly to a server service (WCF) or through the client side interface, including the exposed functionality by the login and change password CCW controls. As mentioned the CCW controls are COM visible and are helpful for programmers of COM client components using the exposed COM interfaces.

1.3 Applicable Documents

ES78001.2620	Functional Requirements Document for proDAS
ES78024.2634	Engineering Specification For User Security System
DB78024.2700	Design Brief For User Security System

1.4 Codes and Standards

ES78001.2620	Functional Requirements Document for proDAS

1.5 Abbreviations and Definitions

Account Mgt GUI	A proDAS application that has been designed specifically for adding, editing, and deleting User Security settings.
CCW	COM-Callable Wrapper
COM	Component Object Model
DB	Design Brief
DCOM	Distributed Component Object Model
DLL	Dynamic Link Library
ES	Engineering Specification
GUI	Graphical User Interface

GUID	Globally Unique IDentifier
ICD	Interface Control Document
IDL	Interface Definition Language
MDS	MDS Aero Support Corporation.
USS	User Security System
WCF	Windows Communication Foundation
WPF	Windows Presentation Foundation

2. DESIGN

2.1 Introduction

2.1.1 Those components necessary to support existing clients are available via COM.

2.1.2 The Login Control is a .NET component with a CCW (COM-Callable wrapper).

2.1.3 The Change Password Control is a .NET component with a CCW (COM-Callable wrapper).

2.1.4 The USS application is implemented as a WPF .NET client application using WCF to communicate with a security server.

2.1.5 The User Security System is registered with installation procedures.

2.1.6 There is no persistent data between clients of any of the components of the User Security System.

2.1.7 The security server database stores all security settings.

2.2 Limited Interface for general users

2.2.1 Interface USS.Client.Common.IUserSystemSecurityClient

2.2.1.1 General

The interface is defined in USS.Client.Common and implemented in USS.Client.

2.2.1.2 Property IsConnected

```
bool IsConnected { get };
```

Determines if there is a connection to the security service.

2.2.1.3 Method Configure

```
void Configure(string endpoint);
```

Argument Name	Description
`Endpoint`	A string that contains information about the server endpoint address to be used by the WCF for communication.

Configures the client to use the specified security service as defined by the endpoint.

2.2.1.4 Method Configure

```
void Configure();
```

Configures the client to use the endpoint specified in the USS.Client.dll.config configuration file (Annex B).

2.2.1.5 Method Dispose

```
void Dispose();
```

Explicitly dispose of the client connection.

2.2.1.6 Method PingServer

```
bool? PingServer();
```

Determines if there is a valid connection to the security service.

The return value can be:

- Null, which means there is no connection to the service;

- False, which means it is connected to the service, but there is a database connection issue;
- True, which means it is connected to the service and the database is available.

2.2.1.7 Method VerifyApplicationUserByPassword

```
ApplicationUserConfirmationData VerifyApplicationUserByPassword(string
applicationName, string userName, SecureString password);
```

Argument Name	**Description**
`applicationName`	An application name.
`userName`	A user name.
`password`	Password for a user (encrypted in the memory - SecureString).

Allows password verification and provides permission details for a given user, application and password. The permission details are returned as a `ApplicationUserConfirmationData` class object (ref. 2.4.1.14).

2.2.1.8 Method VerifyApplicationUserByPassword

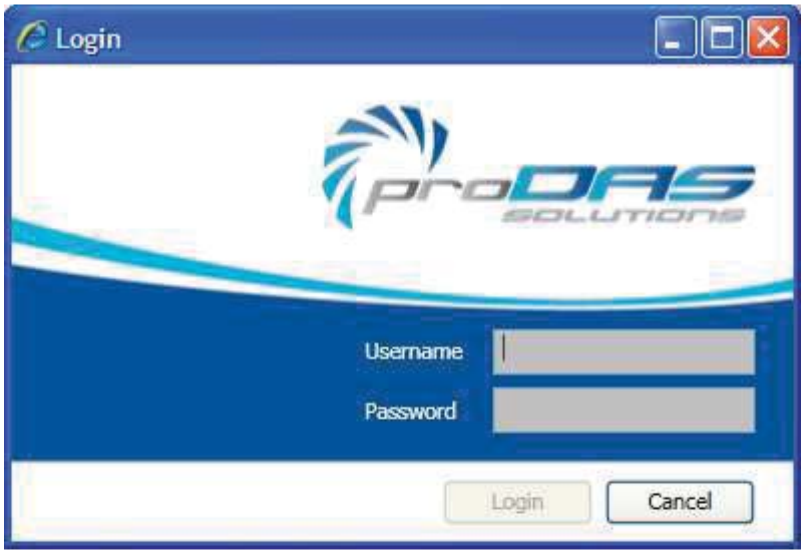
```
ApplicationUserConfirmationData VerifyApplicationUserByPassword(string
applicationName, string userName, string encryptedPassword);
```

Argument Name	**Description**
`applicationName`	An application name.
`userName`	A user name.
`encryptedPassword`	Encrypted password for a user.

Allows password verification and provides permission details for a given user, application and encrypted password. The permission details are returned as a `ApplicationUserConfirmationData` class object (ref. 2.4.1.14).

2.2.2 Interface USS.Client.ILoginControl

2.2.2.1 General



Defined in the USS.Client.LoginControl the ILoginControl serves as a base interface for the LoginControl class which implements the inherited functionality and is also defined in the same DLL. The programmatic identifier (progid) shall be "USS.Client.LoginControl.NET". The following are the two major attributes that the LoginControl class is adorned with in order to be COM visible.

```
[ProgId("USS.Client.LoginControl.NET")]
[Guid("4A3AFD8D-76C7-4D65-9EFF-71CE65E8BBC4")]
```

It implements the ILoginControl and ILoginEvents interfaces.

A WPF client dialog window is wrapped in a .NET CCW.

2.2.2.2 Method ClearUsername

```
bool ClearUsername();
```

Allows the client application to clear the text in the username box.

2.2.2.3 Method ClearPassword

```
bool ClearPassword();
```

Allows the client application to clear the text in the password box.

2.2.2.4 Method ValidatePassword

```
bool ValidatePassword(string userName, string encryptedPassword);
```

Argument Name	**Description**
`userName`	A user name to validate.
`encryptedPassword`	An encrypted password to validate.

Allows a developer to programmatically "validate" an account. The ApplicationName property must be set prior to this method being called. If this is not done, Validate will return a False in all attempts.

2.2.2.5 Property EncryptedPassword

```
string EncryptedPassword { get; }
```

Encrypted password is returned. It is not allowed to set a value to this property. To facilitate command-line logins with the User Security System, client applications must be able to pass the username and password of the logged in user along the command-line to another application.

2.2.2.6 Property ApplicationName

```
string ApplicationName { get; set; }
```

Allows the client application to indicate to the user security system what application it is.

2.2.2.7 Property ExceptionMessage

```
string ExceptionMessage { get; }
```

If an exception or typed SOAP fault is handled during the login then a corresponding exception message is available through this property. Use its value if the LoginFailure event is fired.

2.2.2.8 Property AccessLevel

```
short AccessLevel { get; }
```

Returns a long (0-5) to indicate the login access level. It is not allowed to set a value to this property. Once a user has successfully logged in, the LoginSuccessful Event is raised; the application can then obtain the AccessLevel for this user and this application.

2.2.2.9 Property Username

```
string UserName { get; set; }
```

Returns the text value of the username box. Allows the client application to set the text value of the username text box. To facilitate command-line logins with the User Security System, client applications must be able to pass the username and password of the logged in user along the command-line to another application.

2.2.2.10 Property UsernameLabel

```
string UsernameLabel { get; set;}
```

Returns the caption of the username label. Allows the client application to set the caption value of the username label. The default value is "Username". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.2.11 Property PasswordLabel

```
string PasswordLabel { get; set; }
```

Returns the caption of the password label. Allows the client application to set the caption value of the password label. The default value is "Password". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.2.12 Property LoginButtonText

```
string LoginButtonText { get; set; }
```

Returns the text value of the login button. Allows the client application to set the text value of the login button. The default value is "Login". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.2.13 Property CancelButtonText

```
string CancelButtonText { get; set; }
```

Returns the text value of the cancel button. Allows the client application to set the text value of the cancel button. The default value is "Cancel". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.2.14 Method Show

```
bool Show();
```

Allows the client application to open the LoginControl window as modeless.

2.2.2.15 Method ShowDialog

```
bool ShowDialog();
```

Allows the client application to open the LoginControl window as modal.

2.2.2.16 Method Dispose

```
[DispId(0x60030028)]
bool Dispose();
```

Allows the client application to dispose the LoginControl window. It is highly recommended to dispose the window when it is no longer in use.

2.2.2.17 Property ApplicationSession

```
Guid ApplicationSession { get; }
```

The underlying application session that is created as a part of the login process.

2.2.2.18 Method Initialize

```
bool Initialize(string serviceEndpoint, bool exposeEncryptedPassword,
bool exposeApplicationSession);
```

Initializes the login control with the security server connection information and intended usage.

Argument Name	**Description**
`serviceEndpoint`	The security server connection details.
`exposeEncryptedPassword`	Encrypted password is exposed via the EncryptedPassword property when set true.
`exposeApplicationSession`	Application Session is exposed via the ApplicationSession property when set true.

Basically, the `exposeEncryptedPassword` and `exposeApplicationSession` arguments are mutually exclusive. Below are given the combinations of those two with its intention:

- `exposeEncryptedPassword=true / exposeApplicationSession=false`
  Encrypted password is exposed no sessions calls are made or exposed.
- `exposeEncryptedPassword=false / exposeApplicationSession=true`
  Application Session is exposed but not the encrypted password.
- `exposeEncryptedPassword=false / exposeApplicationSession=false`
  Results in a programmatic interface allowing login validation without a UI.
- `exposeEncryptedPassword=true / exposeApplicationSession=true`
  Not supported.

2.2.2.19 Property FailedLoginAttempts

```
short FailedLoginAttempts { get; }
```

Returns the current failed attempt count. It is not allowed to set a value to this property.

2.2.2.20 Property WindowTitleText

```
string WindowTitleText { get; set; }
```

Allows customization of the window title text. Returns the text value of the window title. Allows the client application to set the text value of the window title. The default value is "Login". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.2.21 Property SessionLengthInMinutes

```
double SessionLengthInMinutes { get; set; }
```

Returns the length of the session. Also allows the client program to define the length of the session in minutes.

2.2.2.22 Method ValidateSession

```
bool ValidateSession(string userName, Guid session);
```

Argument Name	**Description**
`userName`	A user name to validate.

Argument Name	Description
`session`	A session (its GUID) to validate.

Verifies whether the provided username and session are valid. As with ValidatePassword, LoginSuccess and LoginFailure events are fired. The ApplicationName property must be set prior to this method being called. If this is not done, Validate will return a False in all attempts.

2.2.2.23 Method GetResourceDataByPassword

```
Array GetResourceDataByPassword(string user, string encPassword, string
resName, string resType, string resUser);
```

Argument Name	Description
`user`	A user name to validate.
`encPassword`	An encrypted password for the specified user.
`resName`	A resource name.
`resType`	Type of the specified resource.
`resUser`	The name of the resource user.

Verifies first that the username and encrypted password pair is valid and then extracts the resource data for the specified resource name, resource type and resource user.

2.2.2.24 Method Login_VBScript

```
bool Login_VBScript(string serviceEndpoint, bool exposeEncryptedPassword,
bool exposeApplicationSession, string userName, string applicationName,
string wndTitle, string userNameLabel, string passwordLabel);
```

Argument Name	Description
`serviceEndpoint`	The security server connection details.
`exposeEncryptedPassword`	Encrypted password is exposed via the EncryptedPassword property when set true.
`exposeApplicationSession`	Application Session is exposed via the ApplicationSession property when set true.

Argument Name	Description
`userName`	A user name to validate.
`applicationName`	An application name the method is being called from.
`wndTitle`	A caption of the login control window title.
`userNameLabel`	A caption of the username label.
`passwordLabel`	A caption of the password label.

This method is a combination of two methods mentioned above: Initialize and ShowDialog. This is implemented to provide login functionality from a VBScript. An initialization is performed by internally calling the Initialize method, then it assigns the specified captions to the windows title, username and password labels, and finally it calls the ShowDialog method.

2.2.2.25 Method ConfigureClient

```
bool ConfigureClient(string strServiceEndpoint);
```

Argument Name	Description
`serviceEndpoint`	The security server connection details.

This method checks whether the USS client is connected or not, and if it is not , then configures the client, otherwise, confirms that it is connected.

2.2.2.26 Event LoginAttemptsExceeded

```
void LoginAttemptsExceeded();
```

Trigger: Following the third consecutive LoginFailure event being raised, this event is raised.

2.2.2.27 Event CancelButtonClick

```
void CancelButtonClick();
```

Trigger: When the user clicks the cancel button, this event gets raised.

2.2.2.28 Event LoginSuccess

```
void LoginSuccess();
```

Trigger: Upon using the GUI aspect of the Login Control, or by using the Validate method, if the username and password **could** be validated, then the Login Control shall raise a **LoginSuccess** event.

Remarks: Once this event is triggered, the client application should proceed to configure itself based on the AccessLevel.

2.2.2.29 Event LoginFailure

```
void LoginFailure();
```

Trigger: Upon using the GUI aspect of the Login Control, or by using the Validate method, if the username and password **could not** be validated, then the Login Control shall raise a **LoginFailure** event.

2.2.2.30 Usage Conditions and Restrictions

2.2.2.30.1 In all login-scenarios, the client must inform the Login Control of the application that is using the USS. This can be done via the ApplicationName property.

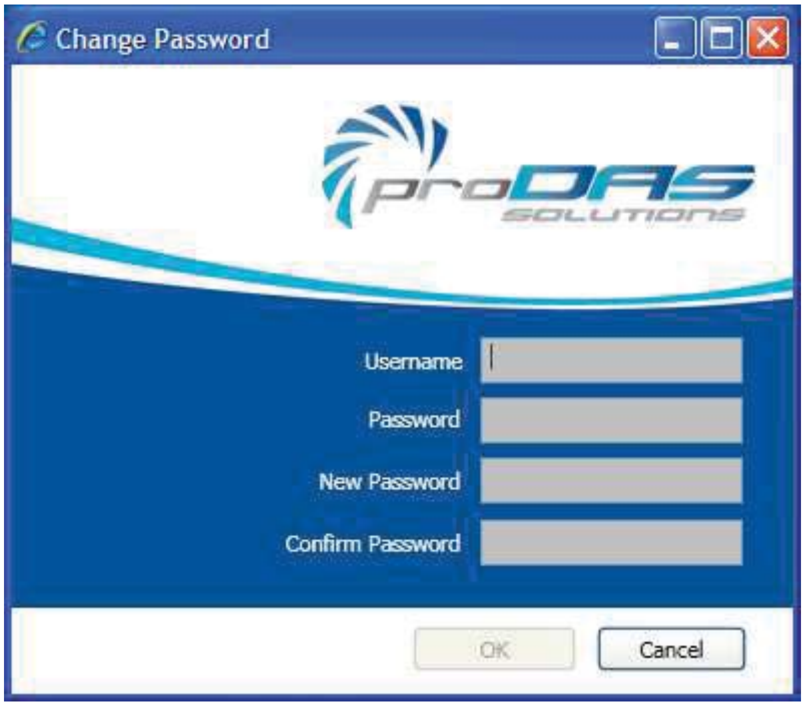
2.2.2.30.2 For a GUI Login, once a user attempts a login, the Login Control will raise either LoginSuccess or LoginFailed events.

2.2.2.30.3 For a Command-Line Login, the username and encrypted password can be passed, and a success value shall be returned. In addition, one of the events (LoginSuccess or LoginFailed) will also be triggered.

2.2.2.30.4 Once a LoginSuccess Event has been triggered, the client may obtain the security permission from the AccessLevel Property. Once obtained, they may configure their application appropriately.

2.2.2.30.5 Check the Annex A for usage examples

2.2.3 **Interface USS.Client.IChangePasswordControl**



2.2.3.1 General

Defined in the USS.Client.ChangePasswordControl the IChangePasswordControl serves as a base interface for the ChangePasswordControl class which implements the inherited functionality and is also defined in the same DLL. The rogrammatic identifier (progid) shall be "USS.Client.ChangePasswordControl.NET". The following are the two major attributes that the ChangePasswordControl class is adorned with in order to be COM visible.

```
[ProgId("USS.Client.ChangePasswordControl.NET")]
[Guid("0F1737C4-9C3B-4AB9-9E27-5CB00DBF3E31")]
```

It implements the IChangePasswordControl and IChangePasswordEvents interfaces.

A WPF .NET client dialog is now wrapped in a .NET CCW.

2.2.3.2 Property EncryptedPassword

```
string EncryptedPassword { get; }
```

Encrypted password is returned. It is not allowed to set a value to this property. To facilitate command-line logins with the User Security System, client applications must be able to pass the username and password of the logged in user along the command-line to another application.

2.2.3.3 Property ExceptionMessage

```
string ExceptionMessage { get; }
```

If an exception or typed SOAP fault is handled during the password change process then a corresponding exception message is available through this property. Use its value if the ServerUnavailable event is fired.

2.2.3.4 Property Username

```
string Username { get; set; }
```

Returns the text value of the username box. Allows the client application to set the text value of the username text box. To facilitate command-line logins with the User Security System, client applications must be able to pass the username and password of the logged in user along the command-line to another application.

2.2.3.5 Property UsernameLabel

```
string UsernameLabel { get; set; }
```

Returns the caption of the username label. Allows the client application to set the caption value of the username label. The default value is "Username". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.3.6 Property PasswordLabel

```
string PasswordLabel { get; set; }
```

Returns the caption of the password label. Allows the client application to set the caption value of the password label. The default value is "Password". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.3.7 Property NewPasswordLabel

```
string NewPasswordLabel { get; set; }
```

Returns the caption of the new password label. Allows the client application to set the caption value of the new password label. The default value is "New Password". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.3.8 Property ConfirmPasswordLabel

```
string ConfirmPasswordLabel { get; set; }
```

Returns the caption of the confirm password label. Allows the client application to set the caption value of the confirm password label. The default value is "Confirm Password". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.3.9 Property CancelButtonText

```
string CancelButtonText { get; set; }
```

Returns the text value of the cancel button. Allows the client application to set the text value of the cancel button. The default value is "Cancel". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.3.10 Property OKButtonText

```
string OKButtonText { get; set; }
```

Returns the text value of the OK button. Allows the client application to set the text value of the OK button. The default value is "OK". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.3.11 Method Show

```
bool Show();
```

Allows the client application to open the ChangePasswordControl window as modeless.

2.2.3.12 Method ShowDialog

```
bool ShowDialog();
```

Allows the client application to open the ChangePasswordControl window as modal.

2.2.3.13 Method Initialize

```
bool Initialize(string serviceEndpoint, bool exposeEncryptedPassword);
```

Initializes the change password control with the security server connection information and intended usage.

Argument Name	Description
`serviceEndpoint`	The security server connection details.
`exposeEncryptedPassword`	Encrypted password is exposed via the EncryptedPassword property when set true.

When using sessions as a credential verification mechanism, note that a user password can be changed outside the scope of any current application sessions.

2.2.3.14 Method Dispose

```
bool Dispose();
```

Allows the client application to dispose the ChangePasswordControl window. It is highly recommended to dispose the window when it is no longer in use.

2.2.3.15 Property WindowTitleText

```
string WindowTitleText { get; set; }
```

Allows customization of the window title text. Returns the text value of the window title. Allows the client application to set the text value of the window title. The default value is "Change Password". To facilitate internationalization and localization, all text labels in the control have been exposed so that developers can change them.

2.2.3.16 Event CancelButtonClick

```
void CancelButtonClick();
```

Trigger: When the user clicks the cancel button, this event gets raised. Developers should use this event to make the Change Password Control go away.

2.2.3.17 Event ServerUnavailable

```
void ServerUnavailable();
```

Trigger: When the control initialises, it will check whether it can connect to the Server. If an error is generated, the control will raise this event.

2.2.3.18 Event ChangePasswordFailure

```
void ChangePasswordFailure();
```

Trigger: Using the GUI only, the user has the ability to change their password. If not successful, the Change Password Control will raise this event.

2.2.3.19 Event ChangePasswordSuccess

```
void ChangePasswordSuccess();
```

Trigger: Using the GUI only, the user has the ability to change their password. If successful, the Change Password Control will raise this event.

2.2.3.20 Usage Conditions and Restrictions

2.2.3.20.1 When this control initializes, it will attempt to figure out if it can connect to the server. If it can't, the control will raise the **ServerUnavailable** Event.

2.3 Check the Annex A for usage examplesInterface for administrative users

2.3.1 USS.Server.IUserSystemSecurity

2.3.1.1 General

The interface is defined and implemented in USS.Server. It defines the methods that will be available for the WCF communication on the server endpoint. In other words, the server service accepts only requests of those method types.

The server itself hosts the WCF service in a managed Windows service.

The current USS prototype has a single call to update all security information, taking in the required data collections. The method that handles that request is called PersistSecurityData, which is considered later in a section below.

In addition, the security token is required from a previous call to logon to the security system.

2.3.1.2 Method Ping

```
bool Ping();
```

Determines if the User Security database is accessible, i.e. if there is a valid connection.

2.3.1.3 Method GetPublicEncryptionKey

```
string GetPublicEncryptionKey();
```

Returns the public encryption key, used for password encryptions.

2.3.1.4 Method PersistSecurityData

```
List<AuditData> PersistSecurityData(Guid session, SecurityDataCollection
data);
```

Argument Name	Description
`session`	A session GUID that was generated when the administrator logged in and a security session got created.
`data`	Consists of the user data, application data, application-user data, resource data and resource-user data collections. Each of those entity data collections represents a list of entity related data, which consists of added, modified and deleted data sub-portions.

This is, perhaps, the most important method, since it does care for security data changes, that is, adding/modifying/deleting users, applications, application-user ties, resources and resource users. The method returns an AuditData class object (ref. 2.4.1.3).

2.3.1.5 Method ChangeUserPassword

```
ChangeUserPasswordConfirmationData ChangeUserPassword(string userName,
string encOldPassword, string encNewPassword);
```

Argument Name	Description
`username`	A user name.
`encOldPassword`	An old encrypted password for the user.
`encNewPassword`	A new encrypted password for the user.

Changes the current password with a new one for the specified user. The method returns a ChangeUserPasswordConfirmationData class object (ref. 2.4.1.16).

2.3.1.6 Method PersistUserUpdateSecurityData

```
List<AuditData> PersistUserUpdateSecurityData(Guid session, UserData
userData);
```

Argument Name	Description
`Session`	A session GUID that was generated when the administrator logged in and a security session got created.
`userData`	Contains the data about a user.

Saves any modifications made to a user data. The method returns a list of AuditData class objects (ref. 2.4.1.3).

2.3.1.7 Method CreateSecuritySession

```
SecuritySessionConfirmationData CreateSecuritySession(string user, string
encPassword);
```

Argument Name	**Description**
`user`	A user name.
`hashedPassword`	Encrypted password for the user.

Creates a security session exclusively for the USS application for a user (the administrator). The user name and password are required. The host session is established as the call is made and the security server creates a time limited session that is tied to the user name, IP and application. Further requests require this session token to be provided. The method returns a SecuritySessionConfirmationData class object (ref. 2.4.1.12).

2.3.1.8 Method ExtendSecuritySession

```
SessionExtensionConfirmationData ExtendSecuritySession(Guid sessionID,
string user, string encPassword);
```

Argument Name	**Description**
`sessionID`	The current session’s GUID.
`user`	A user name.
`hashedPassword`	Encrypted password for the user.

Extends the session for another 60 minutes. This method is designed exclusively for the USS application. All three input parameters are required. The method returns a SessionExtensionConfirmationData class object (ref. 2.4.1.17).

2.3.1.9 Method CreateApplicationSession

```
ApplicationSessionConfirmationData CreateApplicationSession(string user,
string application, string hashedPassword, double
sessionLengthInMinutes);
```

Argument Name	**Description**

Argument Name	Description
`user`	A user name.
`application`	An application name.
`hashedPassword`	Hashed password for the user.
`sessionLengthInMinutes`	The required length in minutes for the session to be created.

Creates an application session for a given user/password expiring after the provided period. Maximum application session length is 4320 minutes (3 days). Later the provided session token can be used for chaining applications. The method returns an ApplicationSessionConfirmationData class object (ref. 2.4.1.13).

2.3.1.10 Method CreateApplicationSessionNoHashConversion

```
ApplicationSessionConfirmationData
CreateApplicationSessionNoHashConversion(string user, string application,
string encPassword, double sessionLengthInMinutes);
```

Argument Name	Description
`User`	A user name.
`application`	An application name.
`hashedPassword`	Encrypted password for the user.
`sessionLengthInMinutes`	The required length in minutes for the session to be created.

This method is the equivalent of the previous CreateApplicationSession method, with the only difference that it accepts already encrypted password instead of hashed password. For more details check the equivalent method (ref. 2.3.1.9). The method returns an ApplicationSessionConfirmationData class object (ref. 2.4.1.13).

2.3.1.11 Method SpawnApplicationSession

```
ApplicationSessionConfirmationData SpawnApplicationSession(Guid session,
string user, string hostApplication, string targetApplication, double
sessionLengthInMinutes);
```

Argument Name	Description
`session`	A session GUID that was generated when a user logged in and an application session got created.
`user`	A user name.
`hostApplication`	A host application name.
`targetApplication`	A target application name.
`sessionLengthInMinutes`	The required length in minutes for the session to be created.

Creates an application session for a given user expiring after the provided period. First it checks that the user has a valid session with the currently hosting application, and if the condition is met then it opens a session for the target application. This is an example of chaining applications using the current session token. Note that a new session token is provided for the target application. The method returns an ApplicationSessionConfirmationData class object (ref. 2.4.1.13).

2.3.1.12 Method VerifyApplicationUserByPassword

```
ApplicationUserConfirmationData VerifyApplicationUserByPassword(string
application, string user, string password);
```

Argument Name	Description
`application`	An application name.
`user`	A user name.
`password`	Hashed password for the user.

Allows password verification and provides permission details for a given user, application and password. The method returns an ApplicationUserConfirmationData class object (ref. 2.4.1.14).

2.3.1.13 Method VerifyApplicationUserBySession

```
ApplicationUserConfirmationData VerifyApplicationUserByPassword(string
application, string user, Guid session);
```

Argument Name	Description

Argument Name	Description
`application`	An application name.
`user`	A user name.
`session`	A session GUID that was generated when a user logged in and an application session got created.

Allows password verification and provides permission details for a given user, application and session. The method returns an ApplicationUserConfirmationData class object (ref. 2.4.1.14).

2.3.1.14 Method ExpireSession

```
ExpireSessionConfiormationData ExpireSession(Guid session, string
userName, string application);
```

Argument Name	Description
`session`	A session GUID that was generated when a user logged in and an application session got created.
`userName`	A user name.
`application`	An application name.

Allows for an early expiration of a session that is no longer required. The method returns an ExpireSessionConfirmationData class object (ref. 2.4.1.15).

2.3.1.15 Method GetUsers

```
List<UserData> GetUsers(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of users; used exclusively for the USS application. For an administration USS session, all users are returned, otherwise only the user associated with the session will be included. The method returns a list of UserData class objects (ref. 2.4.1.8).

2.3.1.16 Method GetUsersByUserCredentials

```
List<UserData> GetUsersByUserCredentials(string application, string user,
string encPassword);
```

Argument Name	Description
`Application`	An application name.
`User`	A user name.
`encPassword`	An encrypted password for the user.

Returns the list of users for the input application. If the input user passes the validation successfully and has the highest permission level for that application then all users registered for that application are returned, otherwise only the user data for the input user will be returned. The method returns a list of UserData class objects (ref. 2.4.1.8).

2.3.1.17 Method GetApplications

```
List<UserData> GetApplications(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of applications; used exclusively for the USS application. The method returns a list of UserData class objects (ref. 2.4.1.8).

2.3.1.18 Method GetResources

```
List<UserData> GetResources(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of resources; used exclusively for the USS application. The method returns a list of UserData class objects (ref. 2.4.1.8).

2.3.1.19 Method GetResourceTypes

```
List<UserData> GetResourceTypes(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of resource types; used exclusively for the USS application. The method returns a list of UserData class objects (ref. 2.4.1.8).

2.3.1.20 Method GetApplicationUsers

```
List<UserData> GetApplicationUsers(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of application users; used exclusively for the USS application. The method returns a list of UserData class objects (ref. 2.4.1.8).

2.3.1.21 Method GetResourceUsers

```
List<UserData> GetResourceUsers(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of resource users; used exclusively for the USS application. The method returns a list of UserData class objects (ref. 2.4.1.8).

2.3.1.22 Method GetAuditData

```
List<AuditData> GetAuditData(Guid sessionID, AuditSearchCriteriaData
searchCriteria);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.
`searchCriteria`	A set of search criteria, such as AuditID, Start/End Dates, Message Content.

Returns audit data by a given set of search criteria; used exclusively for the USS application. The method returns a list of AuditData class objects (ref. 2.4.1.3).

2.3.1.23 Method GetResourceUserDataByPassword

```
ResourceUserConfirmationData GetResourceUserDataByPassword(string user,
string encPassword, string resourceName, string resourceType, string
resourceUserName);
```

Argument Name	**Description**
`user`	A user name.
`encPassword`	Encrypted password for the user.
`resourceName`	A resource name.
`resourceType`	A resource type.
`resourceUserName`	A resource user name.

Returns resource user data for the specified resource name, type and user. But first it validates the user and encrypted password. The method is used exclusively for the USS application. The method returns a ResourceUserConfirmationData class object (ref. 2.4.1.10).

2.3.1.24 Method GetResourceUserDataBySession

```
ResourceUserConfirmationData GetResourceUserDataBySession(Guid session,
string resource, string resourceType, string userName);
```

Argument Name	**Description**
`session`	A session GUID that was generated when a user logged in and a session got created.
`resource`	A resource name.
`resourceType`	A resource type.
`resourceUserName`	A resource user name.

Returns resource user data for the specified resource name, type and user. But first it validates the session. The method is used exclusively for the USS application. The method returns a ResourceUserConfirmationData class object (ref. 2.4.1.10).

2.3.2 Interface USS.Client.Common.IUserSystemSecurityClient

2.3.2.1 General

The interface is defined in USS.Client.Common and implemented in USS.Client.

All methods and properties are available to clients, however some calls require an administration session, i.e. a session associated to the USS application with permission 5 that is not expired.

2.3.2.2 Property IsConnected (ref. 2.2.1.2)

2.3.2.3 Method Configure (ref. 2.2.1.3)

2.3.2.4 Method Configure (ref. 2.2.1.4)

2.3.2.5 Method Dispose (ref. 2.2.1.5)

2.3.2.6 Method ExpireSession

```
ExpireSessionConfirmationData ExpireSession(Guid session, string
userName, string application);
```

Argument Name	**Description**
`session`	A session GUID that was generated when a user logged in and an application session got created.
`userName`	A user name.
`application`	An application name.

Allows for an early expiration of a session that is no longer required. The method returns an ExpireSessionConfirmationData class object (ref. 2.4.1.15).

2.3.2.7 Method PingServer (ref. 2.2.1.6)

2.3.2.8 Method CreateSecuritySession

```
SecuritySessionConfirmationData CreateSecuritySession(string user,
SecureString password);
```

Argument Name	**Description**

Argument Name	Description
`user`	A user name.
`password`	Encrypted password for the user.

Creates a security session exclusively for the USS application. This requires an administration level user that has level 5 permission for the USS application. The method returns a SecuritySessionConfirmationData class object (ref. 2.4.1.12).

2.3.2.9 Method ExtendSecuritySession

```
SessionExtensionConfirmationData ExtendSecuritySession(Guid sessionID,
string user, SecureString password);
```

Argument Name	Description
`sessionID`	The current session GUID that was generated when the current user logged in and the session got created.
`user`	A user name.
`password`	Encrypted password for the user.

Extends the current session for another 60 minutes. The method returns a SessionExtensionConfirmationData class object (ref. 2.4.1.17).

2.3.2.10 Method PersistSecurityData

```
List<AuditData> PersistSecurityData(Guid adminSession,
SecurityDataCollection data);
```

Argument Name	Description
`adminSession`	A session GUID that was generated when the administrator logged in and a security session got created.
`data`	Consists of the user data, application data, application-user data, resource data and resource-user data collections. Each of those entity data collections represents a list of entity related data, which consists of added, modified and deleted data sub-portions.

Persists security data, used exclusively by the USS application and requires an administration session for the USS application. The method returns a list of AuditData class objects (ref. 2.4.1.3).

2.3.2.11 Method ChangeUserPassword

```
ChangeUserPasswordConfirmationData ChangeUserPassword(string user,
SecureString oldPassword, SecureString newPassword);
```

Argument Name	Description
`userName`	A user name.
`encOldPassword`	An old encrypted password for the user.
`encNewPassword`	A new encrypted password for the user.

Changes the current password with a new one for the specified user. The method returns a ChangeUserPasswordConfirmationData class object (ref. 2.4.1.16).

2.3.2.12 Method PersistUserUpdateSecurityData

```
List<AuditData> PersistUserUpdateSecurityData(Guid session, UserData
userData);
```

Argument Name	Description
`session`	A session GUID that was generated when the administrator logged in and a security session got created.
`userData`	Contains the data about a user.

Allows any user to update their password and notes through the USS application. Requires a valid session for the USS application. The method returns a list of AuditData class objects (ref. 2.4.1.3).

2.3.2.13 Method GetUsers

```
List<UserData> GetUsers(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of users; used exclusively for the USS application. For an administation USS session, all users are returned, otherwise only the user associated with the session will be included. The method returns a list of UserData class objects (ref. 2.4.1.8).

2.3.2.14 Method GetUsersByUserCredentials

```
List<UserData> GetUsers(string application, string user, string
encPassword);
```

Argument Name	Description
`application`	An application name.
`user`	A user name.
`encPassword`	An encrypted password for the user.

Returns the list of users for the input application. If the input user passes the validation successfully and has the highest permission level for that application then all users registered for that application are returned, otherwise only the user data for the input user will be returned. The method returns a list of UserData class objects (ref. 2.4.1.8).

2.3.2.15 Method GetApplications

```
List<ApplicationData> GetApplications(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of applications; used exclusively for the USS application. The method returns a list of ApplicationData class objects (ref. 2.4.1.1).

2.3.2.16 Method GetResources

```
List<ResourceData> GetResources(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of resources; used exclusively for the USS application. The method returns a list of ResourceData class objects (ref. 2.4.1.5).

2.3.2.17 Method GetResourceTypes

```
List<ResourceTypeData> GetResourceTypes(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of resource types; used exclusively for the USS application. The method returns a list of ResourceTypeData class objects (ref. 2.4.1.6).

2.3.2.18 Method GetApplicationUsers

```
List<ApplicationUserData> GetApplicationUsers(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of application users; used exclusively for the USS application. The method returns a list of ApplicationUserData class objects (ref. 2.4.1.2).

2.3.2.19 Method GetResourceUsers

```
List<ResourceUserData> GetResourceUsers(Guid sessionID);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.

Returns the list of resource users; used exclusively for the USS application. The method returns a list of ResourceUserData class objects (ref. 2.4.1.7).

2.3.2.20 Method GetAuditData

```
List<AuditData> GetAuditData(Guid sessionID, AuditSearchCriteriaData
searchCriteria);
```

Argument Name	Description
`sessionID`	A session GUID that was generated when a user logged in and a session got created.
`searchCriteria`	A set of search criteria, such as AuditID, Start/End Dates, Message Content.

Returns audit data by a given set of search criteria; used exclusively for the USS application. The method returns a list of AuditData class objects (ref. 2.4.1.3).

2.3.2.21 Method VerifyApplicationUserByPassword (ref. 2.2.1.7)

2.3.2.22 Method VerifyApplicationUserByPassword (ref. 2.2.1.8)

2.3.2.23 Method VerifyApplicationUserBySession

```
ApplicationUserConfirmationData VerifyApplicationUserBySession(string
applicationName, string userName, Guid session);
```

Argument Name	Description
`applicationName`	An application name.
`username`	A user name.
`session`	A session GUID that was generated when a user logged in and an application session got created.

Allows session verification and provides permission details for a given user, application and session. The method returns an ApplicationUserConfirmationData class object (ref. 2.4.1.14).

2.3.2.24 Method CreateApplicationSession

```
ApplicationSessionConfirmationData CreateApplicationSession (string
applicationName, string userName, SecureString password, double
sessionLengthInMinutes);
```

Argument Name	Description
`applicationName`	An application name.
`username`	A user name.
`password`	Password for a user (encrypted in the memory - SecureString).
`sessionLengthInMinutes`	The required length in minutes for the session to be created.

Creates an application session for a given user/password expiring after the provided period. Maximum application session length is 4320 minutes (3 days). The method returns an ApplicationSessionConfirmationData class object (ref. 2.4.1.13).

2.3.2.25 Method GetComputerResourceBySession

```
ComputerResource GetComputerResourceBySession (Guid session, string
resourceName, string userName, string domain, bool
exposePlainTextPassword);
```

Argument Name	**Description**
`session`	A session GUID that was generated when a user logged in and a session got created.
`resourceName`	A resource name.
`userName`	A resource user name.
`domain`	A domain associated with the network credentials.
`exposePlainTextPassword`	Indicator, whether to expose the password or not.

Returns the details of a computer resource. The method returns a ComputerResource class object (ref. 2.4.2.1).

2.3.2.26 Method GetConfigServerResourceBySession

```
ConfigServerResource GetConfigServerResourceBySession (Guid session,
string resourceName, string userName);
```

Argument Name	**Description**
`session`	A session GUID that was generated when a user logged in and a session got created.
`resourceName`	A resource name.
`userName`	A resource user name.

Returns the details of a config server resource. The method returns a ConfigServerResource class object (ref. 2.4.2.2).

2.3.2.27 Method GetDataSourceNameResourceBySession

```
DataSourceNameResource GetDataSourceNameResourceBySession(Guid session,
string resourceName, string userName);
```

Argument Name	**Description**
`session`	A session GUID that was generated when a user logged

Argument Name	Description
	in and a session got created.
`resourceName`	A resource user name.
`userName`	A user name.

Returns the details of a DSN resource. The method returns a DataSourceNameResource class object (ref. 2.4.2.3).

2.3.2.28 Method GetComputerResourceByPassword

```
ComputerResource GetComputerResourceByPassword(string appName, string
appUser, string encPassword, string resourceName, string userName, string
domain, bool exposePlainTextPassword);
```

Argument Name	Description
`appName`	An application name the method is being called from.
`appUser`	An application user (the logged in user).
`encPassword`	An encrypted password for a user.
`resourceName`	A resource name.
`userName`	A resource user name.
`domain`	A domain associated with the network credentials.
`exposePlainTextPassword`	Indicator, whether to expose the password or not.

Returns the details of a computer resource. The method returns a ComputerResource class object (ref. 2.4.2.1).

2.3.2.29 Method GetConfigServerResourceByPassword

```
ConfigServerResource GetConfigServerResourceByPassword(string appName,
string appUser, string encPassword,  string resourceName, string
userName);
```

Argument Name	Description
`appName`	An application name the method is being called from.
`appUser`	An application user (the logged in user).

Argument Name	Description
`encPassword`	An encrypted password for a user.
`resourceName`	A resource name.
`userName`	A resource user name.

Returns the details of a config server resource. The method returns a ConfigServerResource class object (ref. 2.4.2.2).

2.3.2.30 Method GetDataSourceNameResourceByPassword

```
DataSourceNameResource GetDataSourceNameResourceByPassword(string
appName, string appUser, string encPassword, string resourceName, string
userName);
```

Argument Name	Description
`appName`	An application name the method is being called from.
`appUser`	An application user (the logged in user).
`encPassword`	An encrypted password for a user.
`resourceName`	A resource name.
`userName`	A resource user name.

Returns the details of a DSN resource. The method returns a DataSourceNameResource class object (ref. 2.4.2.3).

2.3.2.31 Method Encrypt

```
string Encrypt(SecureString clearText);
```

Argument Name	Description
`clearText`	A secure string to be encrypted.

Encrypts a given secure string.

2.3.2.32 Method Encrypt

```
string Encrypt(string clearText);
```

Argument Name	Description
`clearText`	A regular .Net string to be encrypted.

Encrypts a given .NET string.

2.3.2.33 Method CompareSecureStrings

```
bool CompareSecureStrings(SecureString sec1, SecureString sec2);
```

Argument Name	Description
`sec1`	The first secure string to be compared to the second one.
`Sec2`	The second secure string to be compared to the first one.

Compares two secure strings, true if equal, false otherwise.

2.4 Data Objects

Data objects that are used to return information from the server side calls are defined in USS.Models.dll.

2.4.1 Defined in the USS.Models.dll

2.4.1.1 ApplicationData

Class Member	Type / Return	Name
Property	Guid	ID
Property	String	Name
Property	String	Description
Method	string	ToAuditString

2.4.1.2 ApplicationUserData

Class Member	Type / Return	Name
Property	Guid	ID
Property	Guid	UserID
Property	Guid	ApplicationID

Class Member	Type / Return	Name
Property	short	Permission
Method	string	ToAuditString

2.4.1.3 AuditData

Class Member	Type / Return	Name
Property	Guid	ID
Property	Guid?	SessionID
Property	string	Area
Property	string	Message
Property	bool	IsError
Property	DateTime	OccurredOn

2.4.1.4 AuditSearchCriteriaData

Class Member	Type / Return	Name
Property	Guid?	AuditID
Property	DateTime?	StartDateTime
Property	DateTime?	EndDateTime
Property	string	MessageContents

2.4.1.5 ResourceData

Class Member	Type / Return	Name
Property	Guid	ID
Property	string	Name
Property	Guid	Type
Property	string	Data
Property	string	Description
Method	string	ToAuditString

2.4.1.6 ResourceTypeData

Class Member	Type / Return	Name
Property	Guid	ID
Property	string	Name

2.4.1.7 ResourceUserData

Class Member	Type / Return	Name
Property	Guid	ID
Property	string	UserName
Property	Guid?	UserID
Property	Guid	ResourceID
Property	string	HashedPassword
Property	string	EncryptedPassword
Method	string	ToAuditString

2.4.1.8 UserData

Class Member	Type / Return	Name
Property	Guid	ID
Property	string	Name
Property	string	Notes
Property	string	HashedPassword
Property	string	EncryptedPassword
Property	bool	IsResourceUser
Method	string	ToAuditString

2.4.1.9 SecureCredentialData

Class Member	Type / Return	Name
Property	string	HashedValue
Property	string	EncryptedValue

2.4.1.10 ResourceUserConfirmationData

Base class for client side objects that wrap the data returned from the server GetResourceUserData call.

Class Member	Type / Return	Name
Property	string	Resource
Property	string	Data
Property	string	User
Property	string	EncrypredPassword
Property	bool	ValidResource
Property	string	Password
Property	List<AuditData>	AuditData

2.4.1.11 SecurityDataCollection

Class Member	Type / Return	Name
Property	DataCollection<UserData>	UserData
Property	DataCollection<ApplicationData>	ApplicationData
Property	DataCollection<ApplicationUserData>	ApplicationUserData
Property	DataCollection<ResourceData>	ResourceData
Property	DataCollection<ResourceUserData>	ResourceUserData
Property	int	UserDataChanges

2.4.1.12 SecuritySessionConfirmationData

Class Member	Type / Return	Name
Property	Guid?	Session
Property	bool	ValidUser
Property	short	Permission
Property	List<AuditData>	AuditData

2.4.1.13 ApplicationSessionConfirmationData

Class Member	**Type / Return**	**Name**
Property	Guid?	Session
Property	string	Application
Property	bool	ValidUser
Property	short	Permission
Property	List<AuditData>	AuditData

2.4.1.14 ApplicationUserConfirmationData

Class Member	**Type / Return**	**Name**
Property	string	Application
Property	bool	ValidUser
Property	short	Permission
Property	List<AuditData>	AuditData

2.4.1.15 ExpireSessionConfirmationData

Class Member	**Type / Return**	**Name**
Property	bool	SessionExpired
Property	List<AuditData>	AuditData

2.4.1.16 ChangeUserPasswordConfirmationData

Class Member	**Type / Return**	**Name**
Property	string	UserName
Property	bool	ChangePasswordSuccess
Property	List<AuditData>	AuditData

2.4.1.17 SessionExtensionConfirmationData

Class Member	**Type / Return**	**Name**

Class Member	Type / Return	Name
Property	Guid	Session
Property	bool	IsExtended
Property	List<AuditData>	AuditData

2.4.1.18 USSCustomFault

Class Member	Type / Return	Name
Property	string	DetailedMessage
Property	string	UserMessage
Property	string	DateTime
Property	int	DetailedMessageHashCode
Property	int	FaultOrigin

2.4.2 Defined in the USS.Client.Common.dll

2.4.2.1 ComputerResource (derived from ResourceUserConfirmationData)

Class Member	Type / Return	Name
Property	System.Net.NetworkCredential	Credentials

2.4.2.2 ConfigServerResource (derived from ResourceUserConfirmationData)

2.4.2.3 DataSourceNameResource (derived from ResourceUserConfirmationData)

Class Member	Type / Return	Name
Property	string	DSN

ANNEX A

INTEGRATION EXAMPLES

Integration into a .Net managed code (Login Control and User Verification)

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// The .Net usage code example ------------------------------------------

//////////////////////////////////
using USS.Client.LoginControl;  // USS.Client.LoginControl.DLL
using USS.Client.Common;        // USS.Client.Common.DLL
using USS.Client;               // USS.Client.DLL
//////////////////////////////////


string tmpUserName = string.Empty, tmpPassword = string.Empty; // in case we need
to use
int tmpAccessLevel = 0;

USS.Client.LoginControl.LoginControl loginCtl = null;
const string strEndpoint = @"net.tcp://win7run:2424/USS.Server/USS";

try
{
    loginCtl = new LoginControl();

    loginCtl.CancelButtonClick += new
LoginControl.CancelButtonClickDelegate(loginCtl_CancelButtonClick);
    loginCtl.LoginAttemptsExceeded += new
LoginControl.LoginAttemptsExceededDelegate(loginCtl_LoginAttemptsExceeded);
    loginCtl.LoginFailure += new
LoginControl.LoginFailureDelegate(loginCtl_LoginFailure);
    loginCtl.LoginSuccess += new
LoginControl.LoginSuccessDelegate(loginCtl_LoginSuccess);

    if (loginCtl.Initialize(strEndpoint,    //"null",
//@"net.tcp://win7run:2424/USS.Server/USS",
                            true,           // Use Encrypted password scheme,
                            false           // Do NOT use sessions
                            ))
    {
        loginCtl.Username = string.Empty;
        loginCtl.ApplicationName = "ManagementGUI";
        loginCtl.WindowTitleText = "Login to proDAS Management GUI";
        loginCtl.UsernameLabel = "Username";
        loginCtl.PasswordLabel = "Password";
    }
    else
    {
        MDS.Utilities.Diagnosis.TraceWrite(VerbosityLevel.Highest, "MgtGUI", "Login
Control : Unable to initialize.");
    }

    loginCtl.ShowDialog();

    tmpUserName = loginCtl.Username;
    tmpPassword = loginCtl.EncryptedPassword;
    tmpAccessLevel = loginCtl.AccessLevel;
    //This portion just tests the case when having the raw password we encrypt it
and then validate the username and just encrypted password ///
    USS.Client.Common.IUserSystemSecurityClient client = new
USS.Client.UserSystemSecurityClient();
    if (!client.IsConnected)
        client.Configure(strEndpoint);

    bool isValid =
(client.VerifyApplicationUserByPassword(loginCtl.ApplicationName, tmpUserName,
tmpPassword)).ValidUser;

}
catch
{ }
finally
{
    if (loginCtl != null)
        loginCtl.Dispose();
}

// tmpUserName, tmpPassword, tmpAccessLevel can be used at this point
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Integration into a C++ Unmanaged Code (Login Control)

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// The unmanaged C++ usage code example --------------------------------

////////////////////////////////////////////////////////////////
#import "../../../../../run/bin/OHServer.exe"
#import "../../../../../run/bin/USS.Client.LoginControl.tlb"
using namespace USS_Client_LoginControl; //'.'s replaced by '_'s
////////////////////////////////////////////////////////////////


// Initialize COM.
HRESULT hr = CoInitialize(NULL);

// Create the interface pointer.
ILoginControlPtr pILogCtl(__uuidof(LoginControl));

if (pILogCtl->Initialize("net.tcp://win7run:2424/USS.Server/USS",
                        true, // Use Encrypted password scheme,
                        false // Do NOT use sessions
                                 ))
{
pILogCtl->PutuserName(m_cmdInfo.m_sUserName.AllocSysString());
    pILogCtl->PutApplicationName("MacroEditor");
    pILogCtl->PutWindowTitleText("Login to Macro Editor");
    pILogCtl->PutUsernameLabel("Username");
    pILogCtl->PutPasswordLabel("Password");
}

pILogCtl->ShowDialog();

m_cmdInfo.m_sUserName = (LPCTSTR)pILogCtl->userName;
m_cmdInfo.m_sPassword = (LPCTSTR)pILogCtl->encryptedPassword;
m_lAccessLevel = pILogCtl->AccessLevel;

pILogCtl->Dispose();

// Uninitialize COM.
CoUninitialize();

if (m_lAccessLevel < 1)
{
    AfxMessageBox(IDM_ACCESS_DENIED);
    return FALSE;
}

//Now when calling the m_pConfigServer -> Identify method we can pass the username
and the encrypted
//password (m_cmdInfo.m_sPassword) and the identification should be done against
the encrypted password now.
//The code snippet was taken from LibCfgEds\CfgEdsApp.cpp file (from the
InitInstance method).

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Integration into a VB6 Unmanaged Code (Login Control)

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// The unmanaged VB6 usage code example -------------------------------

////////////////////////////////////////////////////////////////////
// Firstly, the project reference to USS_Client_Login need to be set
////////////////////////////////////////////////////////////////////

Dim loginCtrl As LoginControl
Set loginCtrl = New LoginControl

If loginCtrl.Initialize("net.tcp://win7run:2424/USS.Server/USS", True, False) =
True Then
    loginCtrl.UserName = ""
    loginCtrl.ApplicationName = "RAVE"
    loginCtrl.WindowTitleText = "Login to RAVE"
    loginCtrl.UsernameLabel = "UsernameRAVE"
    loginCtrl.PasswordLabel = "PasswordRAVE"
End If

Call loginCtrl.ShowDialog

If loginCtrl.AccessLevel < 1 Then
    loginCtrl.Dispose
    End
Else
    loginCtrl.Dispose
End If

//The code snippet was taken and modified from RAVE project (in the sobTools-
>Main()).
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Integration into a VB Script (Login Control)

```
'******************************************************************************
'* USSLoginTest.vbs
'******************************************************************************
'*    This script calls the USS Login Control window through COM interface.
'******************************************************************************

Option Explicit


Dim loginCtrl

'******************************************************************************
Err.Number = 0

'Use the ProgID from LoginControl.cs when calling CreateObject method below
Set loginCtrl = CreateObject("USS.Client.LoginControl.NET")

If Err.Number <> 0 Then
    MsgBox "Failed to create the USS Login Control: " & Err.Number
    Err.Number = 0
Else
    Dim res
    res = loginCtrl.Login_VBScript("net.tcp://win7run:2424/USS.Server/USS", True,
False, _
                                       "", "RAVE", "Login 2 RAVE", "Username",
"Password")

       If res Then
              MsgBox "Successfully Logged In"
       Else
              MsgBox "Failed To Login"
       End If
End If

'******************************************************************************
```

Integration into a .Net Managed Code (Change Password Control)

```
USS.Client.ChangePasswordControl.ChangePasswordControl pswdCtl = null;
const string strEndpoint = @"net.tcp://localhost:2424/USS.Server/USS";

try
{
    pswdCtl = new ChangePasswordControl();

    if (pswdCtl.Initialize(strEndpoint, true))
    {
        pswdCtl.Username = "sl1";
        pswdCtl.WindowTitleText = "Change Password";
        pswdCtl.UsernameLabel = "User Name:";
        pswdCtl.PasswordLabel = "Password:";
        pswdCtl.NewPasswordLabel = "New Password:";
        pswdCtl.ConfirmPasswordLabel = "Confirm";

        pswdCtl.ShowDialog();
    }

    //...
}
```

ANNEX B

A USS.CLIENT.DLL.CONFIG FILE SAMPLE

Below is an example of a USS.Client.dll.config configuration file for the USS client applications. Note that the “localhost” in the address attribute’s value needs to be substituted with an IP address or host name of the PC the service host (server) is running.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
      <system.serviceModel>
            <bindings>
                  <netTcpBinding>
                        <binding name="NetTcpLarge" closeTimeout="00:01:00" openTimeout="00:01:00"
                              receiveTimeout="00:01:00" sendTimeout="00:01:00" transactionFlow="false"
                              transferMode="Buffered" transactionProtocol="OleTransactions"
                              hostNameComparisonMode="StrongWildcard" listenBacklog="10"
                              maxBufferPoolSize="524288" maxBufferSize="2147483647" maxConnections="10"
                              maxReceivedMessageSize="2147483647">

                              <readerQuotas maxDepth="2147483647" maxStringContentLength="2147483647"
                              maxArrayLength="2147483647" maxBytesPerRead="2147483647"
                              maxNameTableCharCount="2147483647" />

                              <reliableSession ordered="true" inactivityTimeout="02:00:00" enabled="True" />

                              <security mode="None" />
                        </binding>
                  </netTcpBinding>
            </bindings>

            <client>
                  <endpoint address="net.tcp://localhost:2424/USS.Server/USS" binding="netTcpBinding"
                        bindingConfiguration="NetTcpLarge" contract="Server.IUserSystemSecurity"
                        name="TcpEndPoint">
                  </endpoint>
            </client>
      </system.serviceModel>
</configuration>
```