Oracle® Enterprise Single Sign-On Provisioning Gateway

Command-Line Interface Guide

Release 11.1.2

E27318-02

March 2013

ORACLE®

Oracle Enterprise Single Sign-On Provisioning Gateway Command-Line Interface Guide, Release 11.1.2

E27318-01

Table of Contents

Preface

The *Oracle Enterprise Single Sign-On Provisioning Gateway (Provisioning Gateway) Command Line Interface (CLI) Guide* explains how to use the Provisioning Gateway CLI to configure a provisioning server to send instructions to the Provisioning Gateway Administrative Console.

Audience

This guide is intended for experienced administrators responsible for the planning, implementation, and deployment of Provisioning Gateway. Administrators are expected to understand single sign-on and provisioning concepts, and be familiar with Internet Information Services, Windows Registry settings, and the Oracle Enterprise Single Sign-On and Provisioning Gateway Administrative Consoles. Persons completing the installation and configuration procedure should also be familiar with their company's system standards.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Related Documents

For more information, see the other documents in the Oracle Enterprise Single Sign-On Suite documentation set for this release.

Oracle Enterprise Single Sign-On Suite

Release Notes

Installation Guide

Administrator's Guide

Secure Deployment Guide

User's Guide

Oracle Enterprise Single Sign-On Logon Manager

Deploying Logon Manager with Microsoft Active Directory

Deploying Logon Manager with Microsoft Active Directory Application Mode and Active Directory Lightweight Directory Services

Deploying Logon Manager with a Lightweight Directory Access Protocol Directory

Template Configuration and Diagnostics for Windows Applications

Template Configuration and Diagnostics for Web Applications

Template Configuration and Diagnostics for Mainframe Applications

Oracle Enterprise Single Sign-On Provisioning Gateway

Administrator's Guide

Command Line Interface Guide

Oracle Identity Manager Connector Guide

Sun Java System Identity Manager Connector Guide

IBM Tivoli Identity Manager Connector Guide

Oracle Enterprise Single Sign-On Universal Authentication Manager

Administrator's Guide

User's Guide

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

About the Provisioning Gateway CLIs

The Provisioning Gateway server exposes a Web service interface that allows it to receive instructions submitted to it by any other provisioning server. The Provisioning Gateway CLI is supplied as an integration component for provisioning solutions.

The .NET CLI

The .NET CLI provides an interface for communicating with the Provisioning Gateway Web Service and is installed by default. The programming APIs are kept inside the `Passlogix.Provisioning.dll` assembly, which leverages the main .NET CLI executable as an SDK library. The .NET section of this guide is intended for experienced .NET application programmers responsible for the development of an organization's provisioning solutions.

The Java CLI

The Java CLI exposes several interfaces, a class factory, and supporting types for communicating with the Provisioning Gateway Web Service. These programming APIs are kept inside the class library `pmcli.jar`, which is the same library that is the main executable for the Java CLI and is reused for the SDK. The Java section of this guide describes how to use the interfaces exposed by the Java CLI in your own applications.

This document describes:

- The format of CLI syntax, return values, commands, options, and parameters
- Escaping parameters containing spaces and quotes
- Setting up SSL for the Java CLI
- Examples illustrating the proper usage of CLI commands
- Implementation of the .NET and Java CLIs

The functionality of the .NET and Java CLIs is almost identical. The minor differences are noted throughout the document.

About the Provisioning Gateway CLIs' Installation

The .NET CLI is installed by default. To install the Java CLI, you must select the **Custom** option during installation.

Refer to the *Oracle Enterprise Single Sign-On Suite Installation Guide* for detailed information on installing the Java CLI.

CLI Syntax

The CLI uses the following syntax:

```
usage: pmcli [-url service] [-agent name] [-u login id]

[-p password] [-t date/time] [-f inputfile]

[-security <sec_opts>] "operation"
```

The CLI accepts switches in the following format, in any combination:

Switch	Description
`-arg=value`	Value specified after "="
`-arg` *`value`*	Value specified as next argument
`-arg:`*`value`*	Value specified after ":"
`--arg`	Double dash to start an arg
`/arg`	Forward slash to start an arg
`-u`, `-p`	Equivalent to -security username=<value> password=<value>
`-f`	Executes batch operations from a file, then exits.
`-t`	Alias for -exec. Specifies time to execute provisioning operation.

Differences Between .NET and Java CLI Approaches

The .NET CLI executable is called `pmcli.exe`.

The Java CLI implementation is in a class library called `pmcli.jar`. A batch file, `pmcli.bat`, is provided to execute this library. On Windows, an environment variable, `%PMCLI_ROOT%`, must be set to point to the location where `pmcli.jar` and its supporting libraries reside before executing the batch file. The Java CLI can also be executed manually without the batch file in the following manner:

```
java -cp <classpath> pmcli.Main <args>
```

It might be necessary to edit the `pmcli.bat` file and redefine the d value according to the directions given in the `pmcli.bat` file. The `%P%` value refers to the path where the properties file is stored. The Java CLI can be customized using the properties file. This file must exist along a path without any spaces in the name. By default, the Java CLI is installed on Windows under Program Files, which requires that if you use a properties file, you must set the value of `%P%` to refer to the name of the directory where you will place this file. This directory's name must not contain spaces.

Modes of Operation

There are three supported modes of operation:

- Command-line mode
- Batch mode
- Interactive mode

Command-Line Mode

In this mode, you specify the provisioning operation by entering it on the command line. The following provisioning operations are supported:

Operation	Definition
`ADD_CREDENTIAL`	Add new credential
`MODIFY_CREDENTIAL`	Modify an existing credential
`DELETE_CREDENTIAL`	Delete an existing credential
`DELETE_USER`	Delete SSO user and their stored credentials
`STATUS`	Get status of a pending instruction
`CANCEL`	Cancel a pending provisioning instruction
`EXT_SEARCH`	Search for logon and pending requests
`SET_SETTINGS`	Change the current storage settings
`GET_SETTINGS`	Retrieve the current storage settings
`GET_SCHEMA`	Retrieve the available storage schemas
`CHECK_SERVER`	Check status of server

Each of these operations and their parameters are described in a later section of this document.

If both a batch file and operation are specified on the command line, batch mode takes precedence.

Batch Mode

Batch mode allows you to pass a series of provisioning operations to the CLI in a file specified through the `-f` switch.

Interactive Mode

If there is no operation specified on the command line and no batch file is indicated, the CLI enters interactive mode. In this mode, provisioning operations are specified in a shell-like environment until you enter `quit` or `exit`.

Interactive mode supports three additional commands not available in the command-line or batch mode:

Command	Description
`HELP`	List all commands available
`Help [operation]`	Show syntax for a specific command
`QUIT, EXIT, Q, E`	Exit from interactive mode or stop executing the batch

Smart Defaults

If the url, agent, username, or password switch is not specified, the CLI uses the following defaults:

Switch	Default
`-url`	`http://localhost/v-GO%20PM%20Service/UP.asmx`
`-agent`	The current machine name (on Windows `%MACHINENAME%`).
`-password`	The CLI will prompt for a password.

Difference Between .NET and Java CLI

For security reasons, the .NET CLI obfuscates the password entered by a user (if the user is prompted for a password). For platform-independent reasons, the Java CLI does not obfuscate the password entered by a user.

Operation Execution

When an operation has been executed by the CLI, it outputs the results to the screen. The format output will depend on the operation executed. In general, the result is as follows:

`[RESULT] ID: [GUID]`		
`[RESPONSE]`		
where:		
`[RESULT]`	The result of the provisioning server.	
	`success`	A request was successfully created and placed in the directory. The agent processes this request and marks it either success or failure.
	`noSuchRequest`	The request ID does not exist. This applies to the status and cancel operations.
	`CouldNotCancel`	The request is in a state that does not allow it to be canceled. This applies to the cancel operation.
`[GUID]`	The unique identifier of the provisioning instruction that was submitted successfully.	
`[RESPONSE]`	Additional results returned by the particular provisioning instruction. This applies to the `status`, `ext_search`, `get_settings`, and `get_schema` operations. The results are generally in name-value pair format. This attribute format can be viewed as descriptors for the information being returned.	
In the event of an error, the output will be the exception followed by a descriptive message, as follows: `[exception]: [descriptive error message]`		

Usage

The command `pmcli -?` displays usage and syntax information.

Status Results

When the Logon Manager Agent finishes processing a provisioning instruction, the `Result` attribute of the instruction is set to the result of execution. If the agent fails to process an instruction, the attribute is set to `Failed`, and the `Description` is set to the specific error that occurred. The possible error cases are:

- Failure to decrypt the provisioning instruction.
- Failure to delete the requested instruction.
- Invalid or unknown instruction type.
- Failed to find application specified in instruction.
- Failed to treat modify instruction as an add instruction.
- Failed to add instruction, credential already exists.
- Failed to add instruction, required field not included

Provisioning Operations

The following table lists the specific provisioning operations that can be executed and the specific syntax for each operation:

`add_credential`	Add a new credential for a given user.
`delete_credential`	Delete an existing credential associated with a given user.
`modify_credential`	Modify an existing credential associated with a given user.
`delete_user`	Delete SSO user and their stored credentials.
`status`	Get status of pending and submitted provisioning instructions.
`cancel`	Cancel a pending provisioning instruction.
`ext_search`	Searches for applications, users, and event log entries.
`set_settings`	Change the current storage settings.
`get_settings`	Retrieve the current storage settings.
`get_schema`	Retrieve the available storage schemas.
`check_server`	Checks the status of the server (no errors on success).

Parameters

The operation parameters define the specific characteristics for the request. The set of expected parameters are listed per operation. Each parameter consists of a name-value pair specified as follows:

`sso_userid`	The user's ID as known by Provisioning Gateway. This is the ID that the Provisioning Service uses to locate the user in the Provisioning Gateway data store.
`sso_application`	The name of the application to add a credential to.
`sso_description`	The description of the credential. This field is optional.
`sso_app_userid`	The application's user ID field for this credential.
`sso_password`	The password field for this credential.
`sso_other1`	The third field for this credential.
`sso_other2`	The fourth field for this credential.
`command_id`	The GUID submitted by a successful provisioning request.

SET_SETTINGS

The following describes the specific settings for the `set_settings` operation:

`name`	A comma-delimited list of storage key names.
`value`	A comma-delimited list of storage values.

EXT_SEARCH

The following table defines the specific settings for the `ext_search` operation:

`catalog`	The catalog to search.
`userId`	The sso_userid of the user to find (ext_search).
`logon`	A comma-delimited list of application logon names.
`returnLogons`	Return a list of GUIDs associating stored credential containers to application templates for the selected user.
`returnInstructions`	Return a list of pending instructions.
`uidMatch`	Do an exact or substring match on `userId`.
`startDate`	The start date of the event log.
`endDate`	The end date of the event log.
`eventType`	The type of event to filter the search on.

Syntax

The syntax describes the parameters and format expected for each operation. The following defines each operation and its syntax:

```
ADD_CREDENTIAL sso_userid sso_application [sso_app_userid]
[sso_password] [sso_description] [sso_other1] [sso_other2]

MODIFY_CREDENTIAL sso_userid sso_application sso_app_userid
[sso_description] [sso_password] [sso_other1] [sso_other2]

DELETE_CREDENTIAL sso_userid sso_application
[sso_app_userid] [sso_password] [sso_other1] [sso_other2]

DELETE_USER sso_userid

STATUS sso_userid command_id

CANCEL sso_userid command_id

EXT_SEARCH CATALOG=Applications [userId]

EXT_SEARCH CATALOG=Users [userId] [logon="logon1,logon2,..."]
[returnLogons=true|false] [returnInstructions=true|false]
[uidMatch=substring|equal]
```

If `uidMatch` is not specified, equal is assumed. If `returnLogons` and `returnInstructions` are not specified, false is assumed.

```
EXT_SEARCH CATALOG=EventLog [startDate=mm/dd/yyyy]
[endDate=mm/dd/yyyy] [eventType=amducs]
```

The possible values of `eventType` are:

a	Add Logon
m	Modify Logon
d	Delete Logon
c	Delete User
u	Cancel Request
s	Status Request

These can be used in combination to return matching events.

```
SET_SETTINGS name="key1,key2,..." value="value1,value2,..."
```

Valid keys can be obtained using `GET_SCHEMA`. The number of keys and values must be identical. Each key in the name list is paired with its matching value on the value list (based on position).

GET_SETTINGS	There are no parameters for this command.
GET_SCHEMA	There are no parameters for this command.
CHECK_SERVER	There are no parameters for this command.

Escaping a Comma

Parameters that take comma-delimited values support the "\" (backslash) as an escape character for commas. For example, to enter the value `CN=USERS,DC=DOMAIN,DC=COM` for the `UserPath` in AD, you would issue the following command:

```
SET_SETTINGS name="Storage\AD\UserPath"
value="CN=USERS\,DC=DOMAIN\,DC=COM"
```

Commas that are not escaped are treated as delimiters between multiple values or keys.

Examples

The following examples demonstrate how to use the CLI.

Switches

```
pmcli -username=johns

pmcli -username johns

pmcli -username:johns

pmcli -u:johns

pmcli -u=johns

pmcli -u johns

pmcli /u:johns

pmcli --u:johns
```

The above calls are equivalent and apply to all switches.

Smart Defaults

`pmcli -p:Password`

`url` defaults to `http://localhost/v-go%20pm%20service/up.asmx`

`agent` defaults to machine name

`username` is the current logged on user

`pmcli -u:Administrator -p:Password`

`url` defaults to `http://localhost/v-go%20pm%20service/up.asmx`

`agent` defaults to machine name

`pmcli -url:http://test.com/v-go%20pm%20service/up.asmx -p:mypassword`

`agent` defaults to machine name

`username` is current logged in user

`pmcli`

`url` defaults to `http://localhost/v-go%20pm%20service/up.asmx`

`agent` defaults to machine name

`username` is current logged in user

`password` is prompted (CLI prompts for a password)

Adding a Credential

The following example adds a Lotus Notes credential for the SSO user *joeuser*:

```
pmcli -url "http://example.com/v-GO PM Service/UP.asmx" -agent "PM
Agent" -username=PMAdmin -password=mysecretpassword add_credential
```

```
sso_userid=joeuser sso_application="Lotus Notes"
sso_app_userid=lotususer sso_password=password123 sso_other1=mydomain
```

The first four switches to the CLI indicate:

- The location of the Provisioning Gateway Web service
- The identifier for this agent
- The credentials to use to authenticate against the Web service
- The operation and its parameters.

In this case, the SSO user to provision is *joeuser* and a credential was added for Lotus Notes with credentials of *lotususer* and *password123* in the *mydomain* domain.

Deleting All Credentials for a User

The following example deletes all credentials for the SSO user *joeuser*:

```
pmcli -url "http://example.com/v-GO PM Service/UP.asmx" -agent "PM Agent" -
username=PMAdmin -password=mysecretpassword delete_user sso_userid=joeuser
```

Returning a List of Specific Users

This example returns a list of users with provisioned logons and instructions on the system:

```
pmcli -url "http://example.com/v-GO PM Service/UP.asmx" -agent "PM Agent" -
username=PMAdmin -password=mysecretpassword ext_search catalog=users
returnLogons=true returnInstructions=true
```

Executing Operations from a Batch File

The following example demonstrates how to execute operations from a batch file:

```
pmcli -url:"http://example.com/v-GO PM Service/UP.asmx" -agent:"PM
Agent" -u:PMAdmin -p:mysecretpassword -f=c:\operations.txt
```

The file operations.txt contains one provisioning operation per line:

```
add_credential sso_userid=joeuser sso_application="Lotus Notes" ...

add_credential sso_userid=janeuser sso_application="Lotus Notes" ...

delete_credential sso_userid=jackuser sso_application="Lotus Notes"
```

Running the CLI in Interactive Mode

The following example demonstrates how to run the CLI in interactive mode:

```
pmcli -url:"http://example.pass.com/v-GO PM Service/UP.asmx" -agent:

"PM Agent" -u:PMAdmin -p:mysecretpassword
```

The CLI enters interactive mode and displays the following:

```
Passlogix (R) v-GO PM CLI Version 6.0.0

Copyright (C) Passlogix, Inc. 1998-2005. All rights reserved.

URL: http://example.pass.com/v-GO PM Service/UP.asmx

AGENT: PM Agent”

USERNAME: PMAdmin
```

```
EXECUTE: 10/17/2005-15:07:04

-------------------------------------
```

Type `"e"[xit]` or `"q"[uit]` to end a session.

Displaying Help

```
HELP

HELP [operation]
```

`operation` - Displays help information on that operation.

The user can enter provisioning operations at the prompt similar to the operations in batch mode until he encounters a quit or exit.

Specifying When to Run the Provisioning Operation

The following example demonstrates how to specify when to run the provisioning operation:

Specifying the `-t` switch on the command line followed by a time indicates that the Logon Manager Agent should execute the provisioning operation only on or after the specified time. The operation exists on the directory service and the Provisioning Gateway Agent executes it, but the logon will not be available to the SSO user until the time specified.

The format of `-t` is:

Java: `MM/DD/YYYY-HH:MM:SS`

.NET: `"MM/DD/YYYY HH:MM:SS"`

Using the .NET CLI as an SDK

The Provisioning Gateway .NET CLI must be installed prior to performing the steps in this section. Refer to the *Oracle Enterprise Single Sign-On Suite Installation Guide* for information on installing the Provisioning Gateway .NET CLI.

The .NET CLI is located under `<Passlogix home>\v-GO PM\Client\CLI\DotNet`..

To use the .NET CLI as an SDK, complete the following steps:

1. In your .NET project, add a reference to the `Passlogix.Provisioning.dll`.
2. Create an instance of the IProvisioning interface.
3. Call the available methods on this interface (such as `AddCredential`, etc).
4. Use the returned IProvisioningResult interface to determine success and retrieve results.

Adding a Reference to Passlogix.Provisioning

To add a reference to `Passlogix.Provisioning.dll` in your .NET project:

1. From Visual Studio, load your solution and launch the **Solution Explorer**.
2. Select the applicable .NET project and expand it.
3. Right click on the **References** node and select **Add Reference**.
4. From the dialog, select **Browse** and find `Passlogix.Provisioning.dll` (which you will find under `<Passlogix home>\v-GO PM\Client\DotNet)`.
5. Click **Open**. A new reference to the assembly is created.
6. Open the source file (with `.cs` extension) where the APIs are called, and add the following lines at the beginning of the file:

   ```
   using Passlogix.Provisioning;

   using Passlogix.Provisioning.Exceptions;
   ```

Creating an Instance of the IProvisioning Interface

In the same file, create a method to initialize an instance of the IProvisioning interface and add one of the following lines to that method:

Method 1. If you know the full path

```
IProvisioning iprov =

ProvisioningFactory.CreateFrom(@"<Path to .NET CLI>");
```

Method 2. Load from same directory as provisioning assembly

```
IProvisioning iprov = ProvisioningFactory.CreateFromPrivate();
```

Method 3. To load file from the path (specified by %PATH%)

```
IProvisioning iprov = ProvisioningFactory.CreateFromPath();
```

After you have selected a method for loading, check for errors and then set the credentials for connection to the Provisioning Gateway service.

Use the following code after selecting the loading assembly method:

```
if (iprov != null)
```

```
{

try

{
```

You must first establish a connection to ensure that all resulting calls to the methods do not fail. This method sets credentials for connecting to the provisioning service. It does not actually connect to the service until a provisioning request is made.

There are three ways to connect:

Method 1

```
iprov.Connect("Administrator", "password");
```

Assumes `http://localhost/v-go pm service/up.asmx` and `%COMPUTERNAME%` is the Agent name.

Method 2

Specify the URL and Agent name:

```
iprov.Connect(

"http://<server>/v-go pm service/up.asmx",

"My Agent",

"Administrator", "password");
```

Method 3

Specify the URL:

```
iprov.Connect(

"http://<server>/v-go pm service/up.asmx",

"Administrator", "password");
```

Make provisioning requests via the iprov interface. This method is preferred because the Web service is not local but the user does not necessarily want to specify the agent name (defaults to `%COMPUTERNAME%`). See the sample code section for examples.

```
}

catch (ProvisioningException ex)
{
// Handle exception
}
}
```

After the connection executes successfully, requests can be sent to the Provisioning Gateway Web service through the methods of the `iprov` variable. Each method returns its results in an `IProvisioningnResult` interface. Oracle recommends these methods be called within a `try…catch` block for error handling. Catching the `ProvisioningException` class is sufficient for any exceptions thrown by the CLI. Other exceptions can be handled by adding a catch (`Exception`) block.

Available Methods in iProv Interface

This section lists all the available methods and their parameters for each provisioning operation. The following information is provided for each available method:

- Method name and description
- Method Overload List
- A description of the method's parameters (if applicable)

  One of these parameters requires a special explanation. The options parameter is a dictionary of key-value pairs. The key is the name of the argument used by the CLI on the command line. The value is its value. The developer can set a key-value pair in the dictionary using either the literal name of the key (passed on the command line) or the key constants defined in the `OperationKeys` class.
- Command-line syntax used by the CLI (CLI_Syntax) (if applicable)

  The command-line arguments map directly to the valid keys that can be used to fill the `options` parameter of a method. The `OperationKeys` class has been provided for convenience with constants mapping to the literal value of each key. This can be used to fill or index the options array. For brevity, the CLI Syntax does not show the full syntax. Refer to the Syntax section for full information. The operation name is capitalized. Arguments specified in brackets are optional.

Method	Description
`Connect`	Establishes connection to Web service. This method does not actually attempt the connection but stores the credentials used to connect for use by other methods.

Overload List

```
void Connect(string strUsername, string strPassword);

void Connect(string strURL, string strUsername, string strPassword);

void Connect(

string strURL,

string strAgent,

string strUsername,

string strPassword);
```

Parameter	Description
`strURL`	Web Service URL. Default is http://localhost/v-GO%20PM%20Service/up.asmx
`strAgent`	Identifier for this agent. Default is `%COMPUTERNAME%`.
`strUsername`	Username used to authenticate against the Web service.
`strPassword`	Password used to authenticate against the Web service.

Method	Description
`SetExecTime`	Sets the execution time of the provisioning instruction. This can be used to tell the instruction to execute in the agent at a future date or time after it has been created. If this is not set, it defaults to "Now."

Overload List

```
void SetExecTime(DateTime dtExec);
```

Method	Description
`AddCredential`	Provision the user with a new credential.

Overload List

```
IProvisioningResult AddCredential(
string strUserId,
string strApplication,
string strDescription,
string strAppUserId,
string strPassword);
IProvisioningResult AddCredential(
string strUserId,
string strApplication,
StringDictionary options);
```

Parameter	Description
`strUserId`	User ID of user to be provisioned.
`strApplication`	Name of the application to provision.
`strDescription`	Description of the provisioning instruction.
`strAppUserId`	Application user ID of the credential.
`strPassword`	Password of the credential.
`options`	Hashtable of options (keys specified by `OperationKeys`).

CLI Syntax

```
ADD_CREDENTIAL sso_userid sso_application [sso_app_userid]
sso_password] [sso_description] [sso_other1] [sso_other2]
```

Method	Description
`CancelRequest`	Cancel the provisioning request (before the agent runs).

Overload List

```
IProvisioningResult CancelRequest(string strUserId, string strGuid);
```

Parameter	Description
`strUserId`	User ID of user to be provisioned.
`strGuid`	ID of provisioning instruction to cancel (returned by several methods) that can be canceled.

CLI Syntax

```
CANCEL sso_userid=<username> command_id=<guid>
```

Method	Description
`DeleteCredential`	Delete a provisioned credential.

Overload List

```
IProvisioningResult DeleteCredential(string strUserId,
string strApplication, string strAppUserId, string strOther1,
string strOther2);
IProvisioningResult DeleteCredential(string strUserId,
string strApplication, StringDictionary options);
```

Parameter	Description
`strUserId`	User ID of user to be provisioned.
`strApplication`	Name of the application to provision.
`strAppUserId`	Application User ID of the credential.
`strOther1`	Other field value (1).
`strOther2`	Other field value (2).
`options`	Hashtable of options (keys specified by OperationKeys).

CLI Syntax

```
DELETE_CREDENTIAL sso_userid sso_application [sso_app_userid]
[sso_password] [sso_other1] [sso_other2]
```

Method	Description
`ModifyCredential`	Modify a provisioned credential.

Overload List

```
IProvisioningResult ModifyCredential(string strUserId,
string strApplication, string strAppUserId,
string strDescription, string strPassword, string strOther1,
string strOther2);
IProvisioningResult ModifyCredential(string strUserId,
string strApplication, string strAppUserId,
StringDictionary options);
```

Parameter	Description
`strUserId`	User ID of user to modify.
`strApplication`	Name of the application of credential to modify.
`strAppUserId`	Application User ID of the credential to modify.
`strAppUserId`	Password of the credential to modify.

Method	Description
`strDescription`	Description of the provisioning instruction.
`strOther1`	Other field value (1).
`strOther2`	Other field value (2).
`options`	Hashtable of options (keys specified by OperationKeys).
CLI Syntax	
`MODIFY_CREDENTIAL sso_userid sso_application sso_app_userid [sso_description] [sso_password] [sso_other1] [sso_other2] [sso_password] [sso_other1] [sso_other2]`	

Method	Description
`DeleteUser`	Delete the user container (similar to deleting all credentials for a particular user).
Overload List	
`IProvisioningResult DeleteUser(string strUserId);`	
Parameter	**Parameter**
`strUserId`	User ID of container to delete.
CLI Syntax	
`DELETE_USER sso_userid=<username>`	

Method	Description
`GetStatus`	Ping the server. If it returns successfully without error, the server is functioning.
Overload List `IProvisioningResult GetStatus();`	
CLI Syntax	
`CHECK_SERVER`	

Method	Description
`StatusRequest`	Request the status of a pending provisioning instruction.
Overload List `IProvisioningResult StatusRequest(string strUserId, string strGuid);`	
Parameter	**Parameter**
`strUserId`	User ID to query.
`strGuid`	ID of provisioning instruction (returned by several methods)
CLI Syntax	
`STATUS sso_userid=<username> command_id=<guid>`	

Method	Description
`GetSettings`	Return the directory settings of the PM Web service.
Overload List `IProvisioningResult GetSettings();`	
CLI Syntax	
`GET_SETTINGS`	

Method	Description
`GetSchema`	Get the schema (or list of available options for `SetSettings`).
Overload List `IProvisioningResult GetSchema();`	
CLI Syntax	
`GET_SCHEMA`	

Method	Description
`SetSettings`	Change the settings used by the Web service.
Overload List `IProvisioningResult SetSettings(IDictionary map);`	
Parameter	**Description**
`Map`	Key-value pair for each setting.
CLI Syntax	
`SET_SETTINGS name="key1, key2, ..." value="value1, value2, ..."`	

Method	Description
`ExtSearch`	Search the directory service and return information on users, applications, and logs. This returns a list of applications that can be provisioned for a particular user or all users.

Overload List for Applications

```
IProvisioningResult ExtSearchApplications();
IProvisioningResult ExtSearchApplications(string strUserId);
```

Parameter	Description
`strUserId`	Name of user whose application list should be returned.

Overload List for Users

```
IProvisioningResult ExtSearchUsers();
IProvisioningResult ExtSearchUsers(string strUserId,
StringCollection logons, bool fRetLogons, bool fRetInsts,
bool fMatchExact);
IProvisioningResult ExtSearchUsers(StringDictionary options);
```

Parameter	Description
`strUserId`	User to return information on.
`logons`	Return only these logons (csv format).
`fRetLogons`	Return logon information.
`fRetInsts`	Return pending provisioning instructions.
`fMatchExact`	Use exact match on strUserId.
`options`	Hashtable of options (specified by ExtSearchKeys).

Overload List for Logging

```
IProvisioningResult ExtSearchLog();
IProvisioningResult ExtSearchLog(EventType evt);
IProvisioningResult ExtSearchLog(DateTime dtStart, DateTime dtEnd,
EventType evt);
```

Parameter	Description
`evt`	EventType to return.
`dtStart`	Start date of range to return.
`dtEnd`	End date of range to return.

CLI Syntax

```
EXT_SEARCH CATALOG=Applications [userId=<username>]
EXT_SEARCH CATALOG=Users [userId=<username>]
[logon="logon1,logon2,..."] [returnLogons=true|false]
[returnInstructions=true|false] [uidMatch=substring|equal]
EXT_SEARCH CATALOG=EventLog [startDate=mm/dd/yyyy] [endDate=mm/dd/yyyy]
[eventType=amducs]
```

Retrieving Results

After a provisioning request to the Provisioning Gateway Web Service has completed, an `IProvisioningResult` interface is returned by the called method. Your application can use this interface to determine if the request has completed successfully and retrieve any relevant results. This section shows the available properties on the `IProvisioningResult` interface and how to interpret their values for the methods called from `IProvisioning`.

Interface Definition

```
public interface IProvisioningResult
{
string Response
{
get;
}


bool Success
{
get;
}


string CommandID
{
get;
}


string ErrorMessage
{
get;
}
IDictionary AttributesCollection
{
get;
}
}
```

Property	Description
`Success`	True if the command completed successfully.
`ErrorMessage`	The error string if Success is False. May not always be set.
`CommandID`	The unique ID associated with the completed command (a 32-digit GUID)). All methods except `ExtSearch` return a GUID. However, only the following methods provide a GUID that can be used by the `CancelRequest` and `StatusRequest` operation: • `AddCredential` • `ModifyCredential` • `DeleteCredential`
`Response`	The raw XML response returned by Web service. This is useful if the results need to be re-parsed.
`AttributesCollection`	Detailed results returned by the Web service on Success. The format is a Dictionary of key-value pairs. The methods that fill this property are: • `GetSettings` • `GetSchema` • `StatusRequest` • `ExtSearch`

Attributes Collection

The Attributes Collection is a dictionary of attributes returned by `GetSettings`, `GetSchema`, `ExtSearch`, and `StatusRequest`. The keys are strings that represent the attribute name. The values can refer either to another IDictionary, an IList, or a string. However, types are not mixed within the same collection. After the type is established, the same type is referenced by all keys.

The following table lists the keys and values returned by the provisioning operations and their meanings:

Methods	Description			
`GetSettings`	Returns a collection of string key-value pairs. The key is the name of the setting. The value is its value. These are the storage values set in the registry by the Provisioning Gateway Web Service.			
`StatusRequest`	Returns a collection of string key-value pairs. The *key* is the name of a status property. The *value* is its value. The following status keys are supported:			
	Status Key	**Value**		
	`InstructionState`	PENDING, PROCESSED		
	`Result`	SUCCESS, FAILED		
	`Description`	SUCCESS, <Reason for failure>		
	`Modified`	<Date modified>		
`GetSchema`	The *key* is a string that represents the name of a group of storage settings. The value is an IList. Each IList entry describes one setting under this group. The entry is an IDictionary of string key-value pairs. The key can be one of the following followed by one of the possible values:			
	Key	**Value**		
	`DataType`	Can be string or bool		
	`DisplayDesc`	A description of this setting. Can be empty.		
	`DisplayName`	The friendly name of this setting to display.		
	`Flags`	An internal value used to describe if the settings is non-persistent, must exist.		
	`RegDefault`	The default value for this setting. Can be empty.		
	`RegName`	The name of the registry key.		
	`RegPath`	The relative registry path to this setting.		
	`RegType`	The registry type (DWORD or string).		
	The setting described by this entry becomes a value that can be retrieved or set by `GetSettings` and `SetSettings`.			
`ExtSearch`	Collection of hashtables. (See next section for more information). The key is a string but the type of the returned value depends on the `ExtSearchXXX` called.			

The structure and format of the returned key-value pairs from the AttributesCollection property are designed to closely mirror the console output from the actual CLI. Simply using the CLI will help in understanding the format and structure of the collection returned by these methods.

ExtSearch Results

This section describes the format of the AttributesCollection map returned by ExtSearch.

ExtSearchApplications

Returns:	**.NET:** HashTable of HashTables **Java:** HashMap of HashMaps

Key	**Value**	
Application Name	**HashTable (string key/value pairs)**	
	Key	**Value**
	`HasFourthField`	`True | False`
	`HasPassword`	`True | False`
	`HasThirdField`	`True | False`
	`HasUserId`	`True | False`
	`IsSecurId`	`True | False`
	If IsSecurId is true, then the first four fields are renamed: • `SecurID-UserId` • `SecurID-Other[4th]` • `HasPassword` • `PassKeyType`	

The following are sample search results:

Adobe Acrobat Reader

```
HasFourthField: False

HasPassword: True

HasThirdField: False

IsSecurID: False

HasUserId: False
```

MSN Messenger

```
HasFourthField: False

HasPassword: True

HasThirdField: False

IsSecurID: False

HasUserId: True
```

Visual SourceSafe

```
HasFourthField: False
```

```
HasPassword: True

HasThirdField: True

IsSecurID: False

HasUserId: True
```

ExtSearchUsers

Returns:	**.NET:** HashTable of Lists of HashTables **Java:** HashMap of Lisis of HashMaps

Key	**Value**	
User's Name		
	Logon Entry	
	Key	**Value**
	`name`	Application name
	`modifiedDate`	Date last modified
	`lastUsedDate`	Date last used by SSO
	`Id`	GUID identifier
	Pending Entry	
	`applicationName`	Application
	`createDate`	Date created
	`executeDate`	Date this will execute
	`id`	GUID identifier
	`instructionType`	`ADD` | `MODIFY` | `DELETE`
	`provisioningAgent`	Agent name
	`status`	`SUCCESS` | `Pending`

CLI Output:

```
ext_search catalog=users returnLogons=true
```

This returns a list of logons for all users.

johnd

```
modifiedDate: 2005-08-24 16:43:41Z

lastUsedDate: 2005-08-24 16:43:41Z

name: Adobe Acrobat Reader

id: a75f58c8-a3bd-4d00-bc27-99a587dd98f8


modifiedDate: 2005-08-24 16:43:41Z
```

```
 lastUsedDate: 2005-08-24 16:43:41Z
name: Adobe Acrobat Reader
id: d6bc375d-3f90-400b-a012-6b80aff4ef49

modifiedDate: 2005-09-09 16:28:15Z
lastUsedDate: 2005-09-09 16:28:15Z
name: Visual SourceSafe
id: 80cdc929-61a6-4b86-8763-d5f02b0dbb8b

modifiedDate: 2005-09-01 17:30:26Z
lastUsedDate: 2005-09-01 17:30:26Z
name: Visual SourceSafe
id: 065f5cff-b651-4a3a-a99c-c606059cbad7

modifiedDate: 2005-09-09 16:41:33Z
lastUsedDate: 2005-09-09 16:41:33Z
name: Visual SourceSafe
id: 0a0686b5-3e38-4830-8e02-79b8177de0b4
```

ExtSearchLog

Returns:	**.NET:** HashTable of HashTables **Java:** HashMap of HashMaps

Key	**Value**	
Entry Number	HashTable (string key/value pairs)	
	Key	**Value**
	`applicationName`	Application name
	`eventType`	Type of event (DWORD flag)
	`executeDate`	Date executed
	`id`	GUID identifier
	`provisionedUser`	User provisioned
	`provisioningAgent`	Agent name
	`timeStamp`	Time stamp

CLI Output

```
ext_search catalog=eventLog
```

This returns a list of logons for all users.

Entry 1

```
applicationName:
eventType: 64
executeDate: 0001-01-01 00:00:00.000Z
id: a09b9de7-4b65-464c-8dcb-90219e222991
provisionedUser:
provisioningAgent: SSO PM Console
timestamp: 2005-11-17 18:33:37.290Z
```

Entry 2

```
applicationName:
eventType: 64
executeDate: 0001-01-01 00:00:00.000Z
id: bd444f6c-e3cf-4efc-bbd8-c5e82d55ed96
provisionedUser:
provisioningAgent: SSO PM Console
timestamp: 2005-11-17 18:33:37.370Z
```

Entry 3

```
applicationName:
eventType: 64
executeDate: 0001-01-01 00:00:00.000Z
id: 6eebd1dd-a904-43db-8c22-38ef941e83b3
provisionedUser:
provisioningAgent: SSO PM Console
timestamp: 2005-11-17 18:33:38.960Z
```

Entry 4

```
applicationName: Visual SourceSafe
eventType: 4
executeDate: 2005-11-17 19:28:51.427Z
id: 2c45f078-c9c7-4268-9abd-4e50111ba644
provisionedUser: davidh
provisioningAgent: SSO PM Console
timestamp: 2005-11-17 19:28:51.427Z
```

Sample Code

The following code demonstrates how to call the `AddCredential` method from the `IProvisioning` interface. This example demonstrates adding a credential for the Logon Manager user "johndoe." The application being added is Yahoo and the credentials for this application are "jdoe" and "password." The description of this credential is "Test App."

```
try
{
IProvisioningResult ipr = iprov.AddCredential(
"johndoe",
"Yahoo",
"Test App",
"jdoe",
"password");

//Process results in ipr
if (!ipr.Success)
{
Console.WriteLine(ipr.ErrorMessage);
return;
}

//Display GUID
Console.WriteLine("SUCCESS" + ipr.CommandID);
}
catch (ProvisioningException ex)
{
// Handle Exception...
}
```

Credentials can also be added using an options argument, which is a more flexible method of passing. This method allows the use of additional parameters (some applications require an OTHER1 and OTHER2 field) and their combinations.

The following example demonstrates how to add a credential for the "Visual SourceSafe" application for the SSO user "johndoe." Since this application requires an OTHER1 field, this method is the only way to add the credential.

```
StringDictionary options = new StringDictionary();
options.Add(OperationKeys.DESCRIPTION, "Test App");
options.Add(OperationKeys.APP_USERID, "jdoe");
options.Add(OperationKeys.PASSWORD, "password");

options.Add(OperationKeys.OTHER1, "VGO");

IProvisioningResult ipr = iprov.AddCredential("johndoe",

"Visual SourceSafe", options);
```

Using the Java CLI as an SDK

The Provisioning Gateway CLI must be installed prior to performing the steps in this section. Refer to the *Oracle Enterprise Single Sign-On Suite Installation and Setup Guide* for information on installing the Provisioning Gateway CLI.

The Java CLI is located under `<`*`home directory`*`>\v-GO PM\Client\Java\<`*`version`*`>`.

To use the Java CLI as an SDK, follow these steps:

1. Add `pmcli.jar` and supporting libraries to the CLASSPATH.
2. Import the provisioning classes into your application.
3. Create an instance of the `ProvisioningConnection` class.
4. Create an instance of the `CLIOperationParser` class.
5. Define the operation and operation parameters using a StringMap.
6. Create an instance of the Operation using the object instance created in step 4.
7. Set execution time (otherwise it defaults to "Now").
8. Send Operation instance (step 6) to the Web service using the `ProvisioningConnection` (step 3) instance.
9. Retrieve success and results of operation.

Sample Code

The following code illustrates a simple program that implements each of these steps:

Import these classes into your application:

```
import com.passlogix.vgo.pm.cli.*;
import com.passlogix.vgo.pm.operations.*;
```

Sample routine for calling the web service:

```
void CallWebService(/* Parameters */)
{
```

Arguments to ProvisioningConnection are defined as:

`URL`: the webservice URL
`strAgent`: the user-defined name for the client agent
`strUsername`: the username to connect as
`strPassword`: the password to use for connection

```
ProvisioningConnection conn = new ProvisioningConnection(strURL, strAgent,
strUsername, strPassword);
try
{
```

Begin execution of instruction:

```
CLIOperationParser opParser =
CLIOperationParser.newInstance();
Operation.StringMap options = new Operation.StringMap();
```

Use `OperationKeys` class for most options. Use `ExtSearchKeys` class for ExtSearch operation:

```
options[OperationKeys.USERID] = "davidh";
options[OperationKeys.APPLICATION] = "Visual SourceSafe";
```

strOper can be equal to any operation defined in CLIOperationParser:

```
Operation oper = opParser.parse(strOper, options);
```

Set the execution time of instruction. If you leave the execution time unspecified, it defaults to Now.

```
oper.setExecTime(dtExec);
conn.sendInstruction(oper);
```

Get results if the operation was successful:

```
if (!oper.getSuccess())
{
String strMsg = String.format(
"The command failed: id=%s, msg=%s",
oper.getCommandID(), oper.getError());
return;
}
```

Retrieve command ID and result attributes:

```
String strID = oper.getCommandID());
CollectionsMap map = oper.getResultAttributes());
}
catch (Exception ex)
{ // print exception
}
}
```

For some commands, one or both of these is empty. See the section Using the .NET CLI as an SDK for more information on the command ID and format of result attributes and the available options for each operation. The available operations are defined as static members of the `CLIOperationParser` class. All of the available options and parameters for the supported operations are defined in the OperationKeys and ExtSearchKeys sections of this document.

Class Definitions

The following class definitions show the important constants and methods needed to programmatically send a request to the Provisioning Gateway Web Service.

CLIOperationParser Class

This class inherits from `OperationParser`. An instance of itself can be created by calling `newInstance()`. When an instance exists, it can be used to create Operation objects representing the specific request to be executed on the server:

Following are all supported operations defined as constant strings:

```
static public final String ADD_CREDENTIAL = "add_credential";
static public final String MODIFY_CREDENTIAL = "modify_credential";
static public final String DELETE_CREDENTIAL = "delete_credential";
static public final String DELETE_USER = "delete_user";
static public final String STATUS = "status";
static public final String CANCEL = "cancel";
static public final String CHECK_SERVER = "check_server";
static public final String GET_SETTINGS = "get_settings";
static public final String GET_SCHEMA = "get_schema";
static public final String SET_SETTINGS = "set_settings";
static public final String EXT_SEARCH = "ext_search";
```

To create a new instance of this parser

```
static public CLIOperationParser newInstance();
```

To print the results to an output stream of choice

```
public void printResults(PrintStream out, Operation oper);
```

OperationParser Class

This class is the base class for `CLIOperationParser`. It defines methods for supporting additional operations and creating Operation objects:

To add a new provisioning operation and its support class

```
public void addOperation(String strOper, Class<? extends Operation> c)
```

The supporting class must be derived from the abstract Operation class. This method is intended for internal use.

To create an instance of the Operation object for the given provisioning instruction

```
public Operation parse(String strInstr)
throws InstantiationException, IllegalAccessException
```

This instruction follows the same format as that passed in the command line.

To create an instance of the Operation object based on the operation name:

```
public Operation parseNoOpt(String strOper)
throws InstantiationException, IllegalAccessException
```

To create an instance of the Operation object based on the operation name and its parameters (specified as a map of key/value pairs):

```
public Operation parse(String strOper, Operation.StringMap options)
throws InstantiationException, IllegalAccessException
```

Operation Class

The Operation Class is the base class for all Operations supported by the Java CLI. This class is responsible for constructing the correct message to send to the Web service and for retrieving and storing the response. The following methods can be used to query the results:

Get the raw xml response from the server

```
public String getResponse()
```

Query if the operation executed successfully

```
public boolean getSuccess()
```

Get the GUID associated with this operation after it is executed

```
public String getCommandID()
```

This can be an empty string if no GUID is associated with the operation.

Get any error message if getSuccess returns false.

```
public String getError()
```

Set the execution time of this operation on the server

```
public void setExecTime(Date dtExec)
```

If you do not provide a value for this parameter, the Operation executes immediately. Otherwise the Operation does not execute until the given time.

Get the result attributes array if the operation was successful

```
public CollectionsMap getResultAttributes()
```

An empty CollectionsMap cab be returned if there are no results other than success to return. The format of CollectionsMap is a name/value pair map of lists or other maps. The exact format of which depends on the operation executed. For more information, see the .NET CLI/SDK section.

Execute the operation

```
public String send(ProvisioningConnection conn)
throws PMCLIException, RemoteException
```

You generally should not call this method directly. Instead call:

```
ProvisioningConnection.sendInstruction(...)
```

This passes the Operation object to it.

OperationKeys Interface

The OperationKeys interface defines all the possible parameters that an Operation can accept. The parameters are specified as keys to the StringMap, followed by their value. The exact subset of keys an Operation supports is described in the *Provisioning Gateway* .NET CLI section of this guide:

```
public interface OperationKeys
{
static public final String USERID = "sso_userid";
static public final String APPLICATION = "sso_application";

static public final String DESCRIPTION = "sso_description";
static public final String APP_USERID = "sso_app_userid";
static public final String PASSWORD = "sso_password";
static public final String OTHER1 = "sso_other1";
static public final String OTHER2 = "sso_other2";
static public final String GUID = "command_id";

static public final String NAME = "name";
static public final String VALUE = "value";
}
```

ExtSearchKeys Interface

The ExtSearchKeys interface defines the parameters supported for the ExtSearch operation. The OperationKeys interface does not apply for this operation. Acceptable parameters must come from this list:

```
public interface ExtSearchKeys
{
```

Supported keys for ExtSearch

```
static public final String OPTION_CATALOG = "catalog";
static public final String OPTION_USERID = "userId";
static public final String OPTION_APPLICATION= "applicationName";
static public final String OPTION_EVENTTYPE = "eventType";
static public final String OPTION_STARTDATE = "startDate";
static public final String OPTION_ENDDATE = "endDate";
static public final String OPTION_LOGON = "logon";
static public final String OPTION_SHOWLOGONS = "returnlogons";
static public final String OPTION_SHOWPENDING = "returnInstructions";
static public final String OPTION_UIDMATCH = "uidMatch";
```

Possible values for OPTION_UIDMATCH key

```
static public final String MATCH_EQUAL = "equal";
static public final String MATCH_SUBSTRING = "substring";
```

Possible values for OPTION_CATALOG key

```
static public final String CATALOG_APPS = "Applications";
static public final String CATALOG_EVENTLOG = "EventLog";
static public final String CATALOG_USERS = "Users";
}
```

Setting Up Java for SSL

To set up SSL support for the Java CLI, you must modify a properties file to point to the Java Keystore File root:

1. Download a public version (no private key) of the SSL certificate that will be used. This can be retrieved from the server that is hosting IIS. Save this public certificate as an ssl.cer as follows:
   a. From the server with the SSL certificate, open the Microsoft Management Console by selecting **Start** > **Run**, type **MMC** and click **OK**.
   b. Click **File** > **Add/Remove Certificates Snap-in**. On the **Standalone** tab, click **Add**.
   c. Select the **Certificate** snap-in and click **Add**.
   d. Select **Computer Account** and click **Next**.
   e. Select **Local Computer** and click **Finish**.
   f. Under the **Console Root**, expand **Certificates (Local Computer**).
   g. Expand **Personal** and click **Certificates**.
   h. Right-click the SSL certificate and select **All Tasks** > **Export**.
   i. On the Certificate Export Wizard panel, click **Next**.
   j. On the Export Private Key panel, click **No, do not export the private key**.
   k. Select the file format you want to use (either DER or BASE-64) and click **Next**.
   l. Browse to locate the file you want to export. Click **Next**.
   m. Save as an ssl.cer file.
   n. Click **Finish**, and then click **OK**. This file will be imported into the java keystore on the client (we will create this next).
2. Verify that JDK 1.42+ is installed on the client workstation. There is a binary called `keytool.exe` that you will use to create the keystore.
3. Create a file called `pmcli.jks` with an alias of `pmssl` as follows:
   a. Run: `keytool -import -trustcacerts -file ssl.cer -alias pmssl -keystore pmcli.jks`
   b. Enter a password for the keystore.
   c. When prompted to trust certificate, click **Yes**.
   d. Copy the **pmcli.jks** file to the folder where **pmcli.ja**r is located.
4. Create a `pmcli.properties` file in the folder defined for property files in `pmcli.bat`.
5. Edit `pmcli.properties` by adding the following line:
   ```
   rmi.ssl.trust.keystore.location=pmcli.jks
   ```
   Save the file.
6. Add the full path to the directory where `pmcli.properties` lives (not the full path to the file) to the CLASSPATH.
7. Run `pmcli.bat` and pass an https URL to the `-url` switch.

Enabling SSL does not prevent the CLI from communicating with an http service.