About ESSO-PG .NET CLI SDK

The .NET CLI SDK is provided with Oracle Enterprise Single Sign-On Provisioning Gateway (ESSO-PG). The SDK provides an interface for communicating with the ESSO-PG Web Service. This document describes how to use this interface in your own applications.

Audience

This guide is intended for experienced .NET application programmers responsible for the development of an organization’s provisioning solutions. The SDK provides an interface for communicating with the ESSO-PG Web Service. These programming APIs live inside the assembly Passlogix.Provisioning.dll. This assembly leverages the main .NET CLI executable as an SDK library.

<table>
<thead>
<tr>
<th>Acronym or Abbreviation</th>
<th>Full Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSO Agent</td>
<td>ESSO-LM Agent</td>
</tr>
<tr>
<td>SSO Administrative Console</td>
<td>ESSO-LM Administrative Console</td>
</tr>
<tr>
<td>ESSO-LM</td>
<td>Oracle Enterprise Single Sign-On Logon Manager</td>
</tr>
<tr>
<td>ESSO-AM</td>
<td>Oracle Enterprise Single Sign-On Authentication Manager</td>
</tr>
<tr>
<td>ESSO-KM</td>
<td>Oracle Enterprise Single Sign-On Kiosk Manager</td>
</tr>
<tr>
<td>ESSO-PG</td>
<td>Oracle Enterprise Single Sign-On Provisioning Gateway</td>
</tr>
<tr>
<td>ESSO-PR</td>
<td>Oracle Enterprise Single Sign-On Password Reset</td>
</tr>
<tr>
<td>SSO</td>
<td>ESSO-LM</td>
</tr>
<tr>
<td>FTU</td>
<td>First Time Use</td>
</tr>
<tr>
<td>SSO Agent</td>
<td>ESSO-LM Agent</td>
</tr>
</tbody>
</table>
Installing ESSO-PG .NET CLI

The ESSO-PG.NET CLI must be installed prior to performing the steps in the document. Refer to the *ESSO-PG Installation and Setup Guide* for information on installing the ESSO-PG.NET CLI.

The .NET CLI is located under `<Passlogix home>\v-GO PM\Client\DotNet`. 
Using the .NET CLI as an SDK

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 (i.e. AddCredential, etc).
4. Use the returned IProvisioningResult interface to determine success and retrieve results.

Add a reference to the Passlogix.Provisioning.dll

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 (can be found under <Passlogix home>\v-GO PM\Client\dotnet).
5. Click Open. A new reference to the assembly will be created.
6. Open the source file (with .cs extension) where the APIs will be called, and add the following lines to the top of the file:

```csharp
using Passlogix.Provisioning;
using Passlogix.Provisioning.Exceptions;
```

Create 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:

```csharp
// 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 PM service:

```csharp
// Code to use after method of loading assembly has been selected
if (iprov != null)
{
    try
    {
        // You'll first need to establish a connection
        // or else all resulting calls to the methods will
        // fail. This method sets credentials for connecting
        // to PM service. It does not actually connect to
        // the service until a provisioning request is made.

        // You can connect in three ways:
        iprov.Connect("Administrator", "password");
        // Assumes http://localhost/v-go pm service/up.asmx
        // and %COMPUTERNAME% is the Agent name.

        // Method 2 allows you to specify URL and Agent name
        iprov.Connect(
            "http://<server>/v-go pm service/up.asmx",
            "My Agent",
            "Administrator", "password");

        // Method 3 allows you to specify URL.
        // This method is preferred since the web service
        // is not local but the user does not necessarily
        // want to specify the agent name (defaults to
        // %COMPUTERNAME%).
    }
}
```
iprov.Connect(
    "http://<server>/v-go pm service/up.asmx",
    "Administrator", "password");

// Make provisioning requests via the iprov interface
// Examples of this are given later in this document
...
}
catch (ProvisioningException ex)
{
    // Handle exception
}

After the connection has executed successfully, requests can be sent to the ESSO-PG Web service via the methods of the iprov variable. Each method returns its results in an IProvisioningResult interface. It is recommended 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 IProvisioning 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 ESSO-PG CLI Guide for full syntax information. The operation name is capitalized. Arguments specified in brackets are optional.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Overload List**

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

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strURL</td>
<td>Web Service URL. Default is <a href="http://localhost/v-G0%20PM%20Service/up.asmx">http://localhost/v-G0%20PM%20Service/up.asmx</a></td>
</tr>
<tr>
<td>strAgent</td>
<td>Identifier for this agent. Default is %COMPUTERNAME%.</td>
</tr>
<tr>
<td>strUsername</td>
<td>Username used to authenticate against the Web service.</td>
</tr>
</tbody>
</table>
### strPassword
Password used to authenticate against the Web service.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetExecTime</td>
<td>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”.</td>
</tr>
</tbody>
</table>

#### Overload List

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddCredential</td>
<td>Provision the user with a new credential.</td>
</tr>
</tbody>
</table>

#### Overload List

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

```csharp
IProvisioningResult AddCredential(
    string strUserId,
    string strApplication,
    StringDictionary options);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strUserId</td>
<td>User ID of user to be provisioned.</td>
</tr>
<tr>
<td>strApplication</td>
<td>Name of the application to provision.</td>
</tr>
<tr>
<td>strDescription</td>
<td>Description of the provisioning instruction.</td>
</tr>
<tr>
<td>strAppUserId</td>
<td>Application user ID of the credential.</td>
</tr>
<tr>
<td>strPassword</td>
<td>Password of the credential.</td>
</tr>
<tr>
<td>options</td>
<td>Hashtable of options (keys specified by OperationKeys).</td>
</tr>
</tbody>
</table>

#### CLI Syntax

```
ADD_CREDENTIAL sso_userid sso_application [sso_app_userid] [sso_password] [sso_description] [sso_other1] [sso_other2]
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancelRequest</td>
<td>Cancel the provisioning request (before the agent runs).</td>
</tr>
</tbody>
</table>

**Overload List**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strUserId</td>
<td>User ID of user to be provisioned.</td>
</tr>
<tr>
<td>strGuid</td>
<td>ID of provisioning instruction to cancel (returned by several methods) that can be canceled.</td>
</tr>
</tbody>
</table>

**CLI Syntax**

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeleteCredential</td>
<td>Delete a provisioned credential.</td>
</tr>
</tbody>
</table>

**Overload List**

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

```csharp
IProvisioningResult DeleteCredential(string strUserId, string strApplication, StringDictionary options);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strUserId</td>
<td>User ID of user to be provisioned.</td>
</tr>
<tr>
<td>strApplication</td>
<td>Name of the application to provision.</td>
</tr>
<tr>
<td>strAppUserId</td>
<td>Application User ID of the credential.</td>
</tr>
<tr>
<td>strOther1</td>
<td>Other field value (1).</td>
</tr>
<tr>
<td>strOther2</td>
<td>Other field value (2).</td>
</tr>
<tr>
<td>options</td>
<td>Hashtable of options (keys specified by OperationKeys).</td>
</tr>
</tbody>
</table>

**CLI Syntax**

`DELETE_CREDENTIAL sso_userid sso_application [sso_app_userid] [sso_password] [sso_other1] [sso_other2]`
## Method: ModifyCredential

**Description:** Modify a provisioned credential.

### Overload List

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

### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strUserId</td>
<td>User ID of credential to modify.</td>
</tr>
<tr>
<td>strApplication</td>
<td>Name of the application of credential to modify.</td>
</tr>
<tr>
<td>strAppUserId</td>
<td>Application User ID of credential to modify.</td>
</tr>
<tr>
<td>strPassword</td>
<td>Password of the credential to modify.</td>
</tr>
<tr>
<td>strDescription</td>
<td>Description of the provisioning instruction.</td>
</tr>
<tr>
<td>strOther1</td>
<td>Other field value (1).</td>
</tr>
<tr>
<td>strOther2</td>
<td>Other field value (2).</td>
</tr>
<tr>
<td>options</td>
<td>Hashtable of options (keys specified by OperationKeys).</td>
</tr>
</tbody>
</table>

### CLI Syntax

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

## Method: DeleteUser

**Description:** Delete the user container (similar to deleting all credentials for a particular user).

### Overload List

`IProvisioningResult DeleteUser(string strUserId);`

### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strUserId</td>
<td>User ID of container to delete.</td>
</tr>
</tbody>
</table>

### CLI Syntax

```
DELETE_USER sso_userid=<username>
```
## Using the .NET CLI as an SDK

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetStatus</td>
<td>Ping the server. If it returns successfully without error, the server is functioning.</td>
</tr>
</tbody>
</table>

### Overload List

```csharp
IProvisioningResult GetStatus();
```

### CLI Syntax

```
CHECK_SERVER
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StatusRequest</td>
<td>Request the status of a pending provisioning instruction.</td>
</tr>
</tbody>
</table>

### Overload List

```csharp
IProvisioningResult StatusRequest(string strUserId, string strGuid);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strUserId</td>
<td>User ID to query.</td>
</tr>
<tr>
<td>strGuid</td>
<td>ID of provisioning instruction (returned by several methods)</td>
</tr>
</tbody>
</table>

### CLI Syntax

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSettings</td>
<td>Return the directory settings of the ESSO-PG Web service.</td>
</tr>
</tbody>
</table>

### Overload List

```csharp
IProvisioningResult GetSettings();
```

### CLI Syntax

```
GET_SETTINGS
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSchema</td>
<td>Get the schema (or list of available options for SetSettings).</td>
</tr>
<tr>
<td>SetSettings</td>
<td>Change the settings used by the Web service.</td>
</tr>
<tr>
<td>ExtSearch</td>
<td>Search the directory service and return information on users, applications, logs. This returns a list of applications that can be provisioned for a particular user or all users.</td>
</tr>
</tbody>
</table>

### Overload List

- **SetSettings**

  - IProvisioningResult SetSettings(IDictionary map).

### Parameter Description

- **Map**
  - Key-value pair for each setting.

### CLI Syntax

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

### Overload List for Applications

- **ExtSearchApplications**

  - IProvisioningResult ExtSearchApplications();
  
  - IProvisioningResult ExtSearchApplications(string strUserId);

### Parameter Description

- **strUserId**
  - Name of user whose application list should be returned.

### Overload List for Users

- **ExtSearchUsers**

  - 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.
Using the .NET CLI as an SDK

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fRetInsts</td>
<td>Return pending provisioning instructions.</td>
</tr>
<tr>
<td>fMatchExact</td>
<td>Use exact match on strUserId.</td>
</tr>
<tr>
<td>options</td>
<td>Hashtable of options (specified by ExtSearchKeys).</td>
</tr>
</tbody>
</table>

**Overload List for Logging**

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

**Parameter**

- `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 Using the IProvisioningResult Interface

After a provisioning request to the ESSO-PG Web Service has completed, an `IProvisioningResult` interface is returned by the called method. Your application can use this interface to determine whether 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

```csharp
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 Web service on `Success`. The format is a Dictionary of key-value pairs. The methods that fill this property are:
  - GetSettings
  - GetSchema
  - StatusRequest
  - ExtSearch

**AttributesCollection**

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

The following table describes the meaning of the keys and values returned by the provisioning operations listed:
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GetSettings</strong></td>
<td>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 ESSO-PG Web Service.</td>
</tr>
<tr>
<td><strong>StatusRequest</strong></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><strong>InstructionState</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Result</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Modified</strong></td>
</tr>
<tr>
<td><strong>GetSchema</strong></td>
<td>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:</td>
</tr>
<tr>
<td><strong>ExtSearch</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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, or is hidden. Can be empty (no flags set).
**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 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

<table>
<thead>
<tr>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Name</td>
<td>HashTable (string key/value pairs)</td>
</tr>
<tr>
<td>HasFourthField</td>
<td>True</td>
</tr>
<tr>
<td>HasPassword</td>
<td>True</td>
</tr>
<tr>
<td>HasThirdField</td>
<td>True</td>
</tr>
<tr>
<td>HasUserId</td>
<td>True</td>
</tr>
<tr>
<td>IsSecurId</td>
<td>True</td>
</tr>
</tbody>
</table>

If **IsSecurId** is true, then the first four fields are renamed:

- SecurID-UserId
- SecurID-Other[4th]
- HasPassword
- PassKeyType

**CLI Output:**

```
ext_search catalog=applications userid=<username>
```

This returns a list of supported applications for the named user.
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 Lists of HashMaps

<table>
<thead>
<tr>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>User’s name</td>
<td>List of HashMaps. Each entry can be either a logon or pending provisioning instruction.</td>
</tr>
</tbody>
</table>

**Logon entry**

<table>
<thead>
<tr>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Application name</td>
</tr>
<tr>
<td>modifiedDate</td>
<td>Date last modified</td>
</tr>
<tr>
<td>lastUsedData</td>
<td>Date last used by SSO</td>
</tr>
<tr>
<td>Id</td>
<td>GUID identifier</td>
</tr>
</tbody>
</table>

**Pending entry**

<table>
<thead>
<tr>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>Application name</td>
</tr>
<tr>
<td>createDate</td>
<td>Date created</td>
</tr>
<tr>
<td>executeDate</td>
<td>Date this will execute</td>
</tr>
<tr>
<td>id</td>
<td>GUID identifier</td>
</tr>
<tr>
<td>instructionType</td>
<td>ADD</td>
</tr>
<tr>
<td>provisioningAgent</td>
<td>Agent name</td>
</tr>
<tr>
<td>status</td>
<td>SUCCESS</td>
</tr>
</tbody>
</table>

**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: 80cd929-61a6-4b86-8763-d5f02b0dbb8b

modifiedDate: 2005-09-01 17:30:26Z
lastUsedDate: 2005-09-01 17:30:26Z
name: Visual SourceSafe
id: 065f5cfe-b651-4a3a-a99c-c606059c5bad7

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

<table>
<thead>
<tr>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry Number</td>
<td>HashTable (string key/value pairs)</td>
</tr>
<tr>
<td>KEY</td>
<td>VALUE</td>
</tr>
<tr>
<td>applicationName</td>
<td>Application name</td>
</tr>
<tr>
<td>eventType</td>
<td>Type of event (DWORD flag)</td>
</tr>
<tr>
<td>executeDate</td>
<td>Date executed</td>
</tr>
<tr>
<td>id</td>
<td>GUID identifier</td>
</tr>
<tr>
<td>provisionedUser</td>
<td>User provisioned</td>
</tr>
<tr>
<td>provisioningAgent</td>
<td>Agent name</td>
</tr>
<tr>
<td>timeStamp</td>
<td>Time stamp</td>
</tr>
</tbody>
</table>

**CLI Output:**

ext_search catalog=eventLog

This returns a list of log entries.
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
Sample Code (AddCredential)
The following code demonstrates how to call the AddCredential method from the IProvisioning interface. This example demonstrates adding a credential for the ESSO-LM 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”.

```csharp
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:

```csharp
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);
```

This example demonstrates how to add a credential for the “Visual SourceSafe” application for the ESSO-LM user “johndoe”. Since this application requires an OTHER1 field, this method is the only way to add the credential.