Oracle® Enterprise Single Sign-on Logon Manager

How-To: Understanding the ESSO-LM Event Notification API

Release 11.1.1.2.0

20414-01

Oracle Enterprise Single Sign-on Logon Manager How-To: Understanding the ESSO-LM Event Notification API

Release 11.1.1.2.0

20414-01

Table of Contents

Table of Contents 3

Introduction 4

  About This Guide 4

  Prerequisites 4

  Terms and Abbreviations 4

  Accessing ESSO-LM Documentation 4

Understanding the ESSO-LM Notification Service API 5

  Overview 5

    Event Handling Tasks 5

    The `SSONotificationService` Co-Class 6

  Sending Data (Producer) 6

    Producer Identification 6

    Event Notification 7

    Security Measures 8

  Receiving Data (Consumer) 8

    Receiving Data in a “Push” Model 8

    Receiving Data in a “Pull” Model 10

Introduction

About This Guide

This document describes the ESSO-LM Event Notification Service API. The service allows the sending and receiving of event-based information to and from Oracle ESSO applications. This guide is intended to aid developers writing third-party software that needs to exchange data with one or more Oracle ESSO applications.

Prerequisites

Readers of this document should have a thorough understanding of software development using the Microsoft .NET framework, including the Component Object Model (COM) technology, and related concepts.

Terms and Abbreviations

The following table describes the terms and abbreviations used throughout this guide:

Term or Abbreviation	Description
ESSO-LM	Enterprise Single Sign-On Logon Manager
Agent	ESSO-LM client-side software
Console	ESSO-LM Administrative Console

Accessing ESSO-LM Documentation

We continually strive to keep ESSO-LM documentation accurate and up to date. For the latest version of this and other ESSO-LM documents, visit http://download.oracle.com/docs/cd/E15624_01/index.htm.

Understanding the ESSO-LM Notification Service API

Overview

The ESSO-LM Notification Service (referred to as "the service" for the remainder of this document) allows the sending and receiving of event data between Oracle ESSO applications. The service runs as a Windows system service and acts as a global events repository and an event router.

The service runs as a Windows system service and distinguishes between the following application roles:

- **Producer** – an application that sends events to other applications
- **Consumer** – an application that receives events from other applications

Event Handling Tasks

The service handles events as follows:

- **Store events received from producers.** The service enumerates and retains 1000 latest events received for each producer and each running session. Once the event buffer is full, the oldest event is discarded for each new event that enters the buffer. Each event can be uniquely identified by producer GUID, session GUID, and its consecutive position in the buffer.
- **Transmit events to consumers.** The service uses the following interface to transmit events:

```
[
        object,
        uuid(DD9E48CA-63D2-4106-876D-4DDEAA063B6F),
        dual,
        nonextensible,
        helpstring("Allows Consumers to access to the information about event"),
        pointer_default(unique)
]
interface ISSONotificationEvent: IDispatch
{
        [propget, id(1), helpstring("Gets event order number")]
        HRESULT Number([out, retval] ULONG* pVal);

        [propget, id(2), helpstring("Gets notification event code")]
        HRESULT NotificationCode([out, retval] ULONG* pVal);

        [propget, id(3), helpstring("Gets progress value")]
        HRESULT Progress([out, retval] LONG* pVal);

        [propget, id(4), helpstring("Gets event importance level")]
        HRESULT Level([out, retval] ULONG* pVal);
```

```
            [propget, id(5), helpstring("Gets additional data")]
                    HRESULT AdditionalData([out, retval] BSTR* pVal);

                    [propget, id(6), helpstring("Gets event time")]
                    HRESULT Time([out, retval] DATE* pVal);
            };
```

The `SSONotificationService` Co-Class

The following IDL code describes the service's co-class used by producers and consumers:

```
        [
                uuid(FBB13217-02AB-42DF-8867-69B8DD935BA9),
                helpstring("SSO Notification Service class")
        ]
        coclass SSONotificationService
        {
                // Allows Consumers to subscribe for event notifications:
                [default] interface ISSONotificationService;
                // Allows Consumers to access to the information about events:
                interface ISSONotificationEventReader;
                // Allows Producers to obtain ISSONotificationEventWriter pointer for event
        raising:
                interface ISSOWriterManager;
        };
```

Sending Data (Producer)

Producers should follow the guidelines below to properly interface with the service.

Producer Identification

A producer must implement the `ISSOProducerInfo` interface to uniquely identify itself to the service:

```
        [
                object,
                uuid(4961B340-D358-4A0E-B8FB-6E2A4BF2DFDD),
                dual,
                nonextensible,
                helpstring("Provides information about Producer"),
                pointer_default(unique)
        ]
        interface ISSOProducerInfo : IDispatch
        {
                [propget, id(1), helpstring("Gets Terminal Services session identifier")]
                HRESULT SessionId([out, retval] ULONG* pVal);

                [propget, id(2), helpstring("Gets Producer GUID")]
                HRESULT ProducerGuid([out, retval] BSTR* pVal);

                [propget, id(3), helpstring("Gets Producer description")]
                HRESULT ProducerDescription([out, retval] BSTR* pVal);
        };
```

Event Notification

When an event occurs, the producer passes the event data to the service via the `ISSONotificationEventWriter` COM interface:

```
        [
                object,
                uuid(72A23F33-927D-4e01-8B50-759262519076),
                dual,
                nonextensible,
                helpstring("Allows Producers to raise new events"),
                pointer_default(unique)
        ]
        interface ISSONotificationEventWriter : IDispatch
        {
                [id(1), helpstring("Raises new event")]
                HRESULT AddEvent([in] ULONG nNotificationCode, [in] LONG nProgress, [in] ULONG
        nLevel, [in] BSTR sAdditionalData);
        };
```

To obtain a pointer to this interface, the producer must implement the `ISSOProducerInfo` interface mentioned earlier and pass its pointer into the `GetWriter` method of the service's `ISSOWriterManager` interface shown below:

```
        [
                object,
                uuid(4490B430-81FD-48f5-BCD9-F9F0A82C6832),
                dual,
                nonextensible,
                helpstring("Allows Producers to obtain ISSONotificationEventWriter pointer for
        event raising"),
                pointer_default(unique)
        ]
        interface ISSOWriterManager : IDispatch
        {
                [id(1), helpstring("Returns ISSONotificationEventWriter pointer for specified
        Producer")]
                HRESULT GetWriter([in] IDispatch* pProducerInfo, [out,retval] IDispatch**
        pEventWriter);
        };
```

Security Measures

The service only accepts events from producers whose executables have been signed by Oracle.

A producer requesting a pointer to the `ISSONotificationEventWriter` using the `ISSOWriterManager::GetWriter` method is validated as follows:

1. The producer's process identifier (PID) is obtained (based on the producer's `ISSOProducerInfo` data passed into the method via the `CoGetServerPID` function).
2. The signature of the producer executable corresponding to the retrieved PID is checked against the information stored in the Windows registry or through the COM Security Initialization process.

> **Note:** The service cannot guarantee a valid signature check when the producer executable is remote.

Additionally, Oracle highly recommends that producers and consumers validate the service's signature as follows:

1. Obtain the service's PID using the `CoGetServerPID` function from one of the `ISSONotificationService` sub-interfaces (`ISSONotificationEventReader`, `ISSOWriterManager`, `ISSONotificationEventWriter`, or `ISSONotificationEvent`).
2. Check the signature of the executable corresponding to the retrieved PID.

Receiving Data (Consumer)

Consumers can receive data using either the "push" or "pull" model.

Receiving Data in a "Push" Model

In the "push" model, consumers must do the following to receive event data:

1. Implement the `_ISSONotificationServiceEvents` interface to handle events:

```
[
        uuid(88AD71A0-0A9A-4916-BE26-E82C4F41BF3F),
        helpstring("Sink interface to handle events")
]
dispinterface _ISSONotificationServiceEvents
{
  properties:
  methods:
        [id(1), helpstring("Handles notification event")]
        HRESULT HandleEvent([in] IDispatch* pEvent);
};
```

The `pEvent` parameter referenced above stores the pointer to the object implementing the `ISSONotificationEvent` and `ISSOProducerInfo` interfaces described earlier:

```
        [
                uuid(C8DCA6F1-2009-4A04-9E4C-BA7CB4CBA86C),
                helpstring("SSO Event class")
        ]
        coclass SSONotificationEvent
        {
                [default] interface ISSONotificationEvent;
                interface ISSOProducerInfo;
        };
```

2. Subscribe to the service event stream by passing the `_ISSONotificationServiceEvents` event handler interface into the method of the `ISSONotificationService` interface:

```
        [
            object,
            uuid(079F0093-99CB-4FCF-900E-18DAD87ED316),
            dual,
            nonextensible,
            helpstring("Allows Consumers to subscribe and unsubscribe for events"),
            pointer_default(unique)
        ]
        interface ISSONotificationService : IDispatch
        {
            [id(1),
            helpstring("Subscribes event handler to events from specified producer and user
and returns subscription cookie")]
            HRESULT SubscribeToEvents([in] ULONG nSessionId, [in] BSTR sProducerGuid, [in]
IUnknown* pEventHandler, [out,retval] ULONG* pCookie);

            [id(2),
            helpstring("Unsubscribes event handler from events from specified producer and
user using cookie returned by SubscribeToEvents method")]
            HRESULT UnsubscribeFromEvents([in] ULONG nSessionId, [in] BSTR sProducerGuid, [in]
ULONG nCookie);
        };
```

When a new event arrives, the service transmits the event data to all subscribed consumers.

Receiving Data in a “Pull” Model

In the “pull” model, a consumer receives the latest events from a producer using the service’s `ISSONotificationEventReader` interface:

```
[
        object,
        uuid(5C4C57D9-D0B1-46AC-A45C-E41C55A7FEF8),
        dual,
        nonextensible,
        helpstring("Allows Consumers to get the information about latest events"),
        pointer_default(unique)
]
interface ISSONotificationEventReader : IDispatch
{
        [id(1), helpstring("Gets the latest event from specified producer and user")]
        HRESULT GetLastEvent([in] ULONG nSessionId, [in] BSTR sProducerGuid, [out, retval]
IDispatch** pVal);

        [id(2), helpstring("Returns array containing specified number of latest events
from specified producer and user")]
        HRESULT GetLatestEventsList([in] ULONG nSessionId, [in] BSTR sProducerGuid, [in]
ULONG nCount, [out, retval] VARIANT* eventsArray);
};
```

The service returns event data as pointer (or a safe array of pointers) to the implementations of the `ISSONotificationEvent` interface described earlier.