Tech Note 804
Consuming and Acknowledging Alarms Using Application Server QuickScript

Topic#: 002590
Created: October 2011

Introduction

This *Tech Note* provides detailed instructions in three parts:

- **Consuming Alarms Using Application Server QuickScript**
- **Acknowledging Alarms Using Application Server QuickScript**
- **Testing Application Server QuickScript for Consuming and Acknowledging Alarms**

This is done by first creating an Instance of $UserDefined Template Object, and then creating 2 different scripts – one for consuming Alarms and second for Acknowledging Alarm. Each Script is further documented with code snippets. By following along this TechNote, you will have a $UserDefined Instance created and deployed in the Galaxy. Further, at the end of it, you will test both the Scripts to Consume and Acknowledge Alarm. The end result will be Alarm Records logged in the Logger, depending on the Alarm Query.

This *Tech Note* provides the following benefits:

- It provides a way to consume Galaxy Alarms, so that Alarms can be presented in a custom format if desired.
- It serves as an example how to use Methods of a COM dll in Application Server Quickscript.

Application Versions

- Wonderware Application Server 3.1 and later
- Alarm Toolkit 8.0

Note: For this *Tech Note*, WAS 3.1 SP3 P01 and MS Windows XP SP3 were used.

About the Alarm Toolkit

The Wonderware Alarm Toolkit can be used to create custom Alarm Provider and custom Alarm Consumer. A custom Alarm Provider and Alarm Consumer typically uses the COM wrappers like **wnwrapserver.dll** and **wnwrapconsumer.dll** to provide and consume Alarms. These COM wrappers (dlls) are installed by InTouch® or Application Server Installation in **...Program Files\Common Files\ArchestrA** folder.

The current version of Alarm Toolkit is 8.0 (November 2011). The Alarm Toolkit installation CD is available within the **Toolkits** CD of Advanced Development Studio. Alarm Toolkit 8.0 is supported with InTouch 8.0 and higher and with Application Server 2.0 and higher. This is because the wrappers for Alarm Toolkit are installed by the product installations of InTouch and Application Server.

For more details on the Alarm Toolkit, refer to the **AlarmToolKit User Guide on WDN**.

Prerequisites

Before diving into specifics of each script, there are couple of setup tasks required to be able to use the Alarm Toolkit COM Wrapper. In this example, wnwrapconsumer.dll will be used since we are interested in consuming Alarms. Secondly, you need to create a $UserDefined Object.

Import wnwrapConsumer.dll

1. Open the ArchestrA IDE.
2. Click **Import/Script Function Library** and import from **C:\Program Files\Common Files\ArchestrA\wnwrapConsumer.dll**.

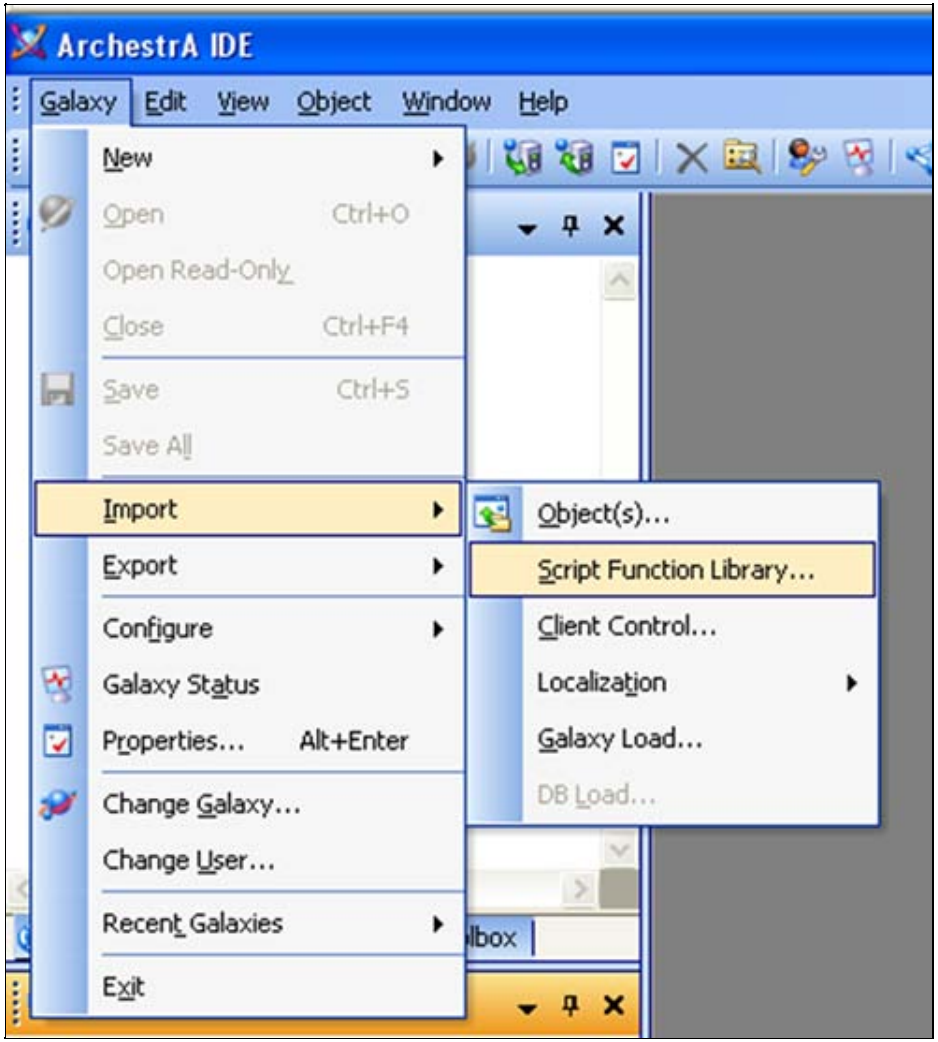

FIGURE 1: GALAXY -> IMPORT -> SCRIPT FUNCTION LIBRARY

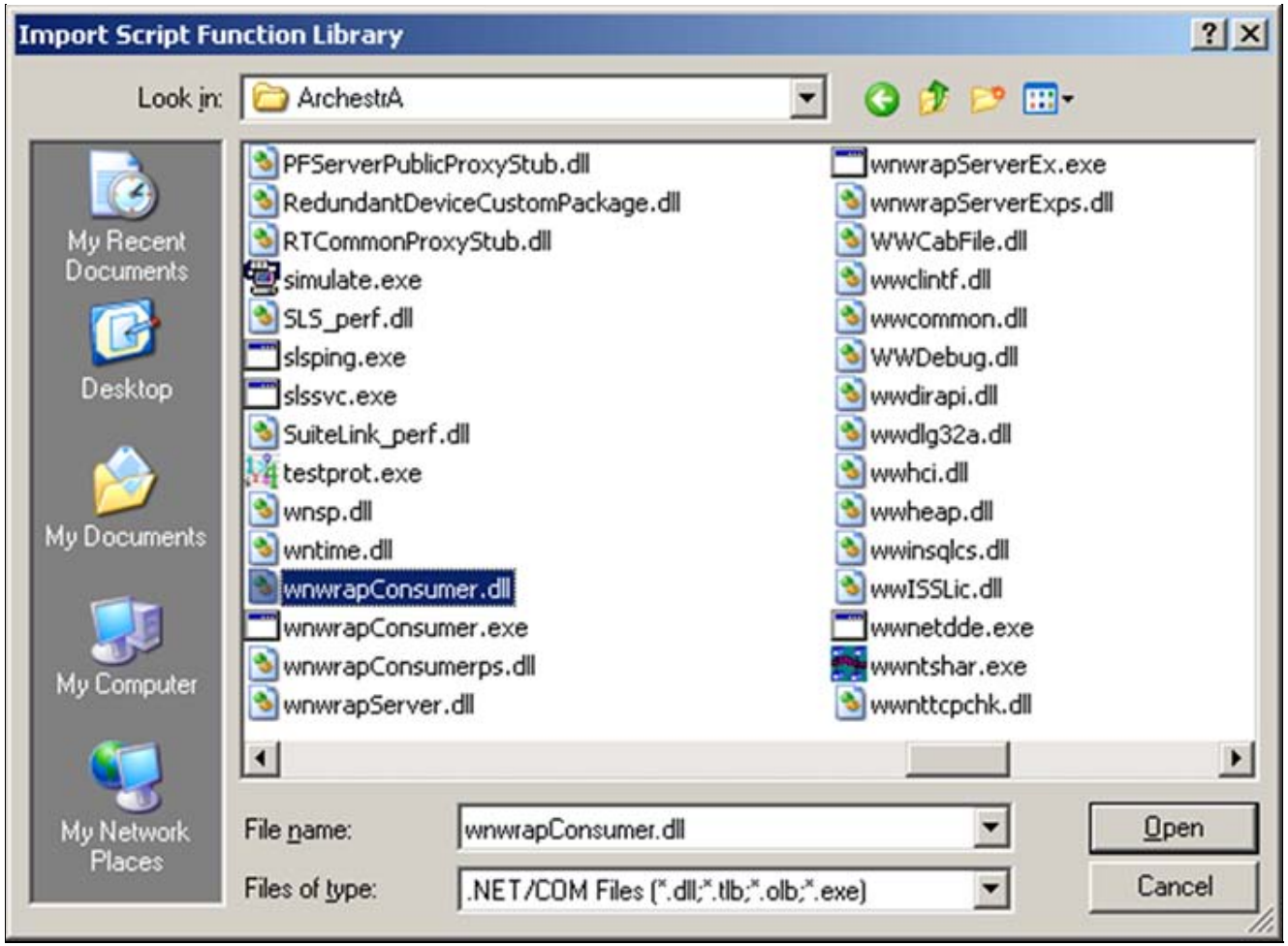

FIGURE 2: FILE LOCATION

FIGURE 3: IMPORT SUCCEEDED

After importing:

- The **wnwrapConsumer.dll** is copied to following folder: **C:\Program Files\ArchestrA\Framework\FileRepository\\Vendors\ArchestrA\wnwrapConsumer.dll**.

Create an Instance of $UserDefined

1. Create an Instance of the **$UserDefined Object**.

   For this *Tech Note* it is called **UD_AlmConsumer**. Host the UD_AlmConsumer under **Area_001**.

2. Proceed to **Consuming Alarms Using Application Server QuickScript**.

Consuming Alarms Using Application Server QuickScript

Create UDAs

1. Open the **UD_AlmConsumer** object instance in ArchestrA IDE and select the UDAs tab.
2. Create following UDAs that will be used in the script:

- **Booleans: bConsumeAlms**: Used to trigger the script for consuming and logging Alarm Records.

Create a Script

- Click the **Scripts** tab, then **Add Script**. Call the script **AlmCons**. This script is used to Consume Alarms and then Log those Alarm Records in Logger.

Verify that the Alarm Consumer Class Methods are Available for Use in the Script

- Use the **Display Script Function Browser** button at the far right of the window (Figure 4 below).

FIGURE 4: DISPLAY SCRIPT FUNCTION BROWSER BUTTON

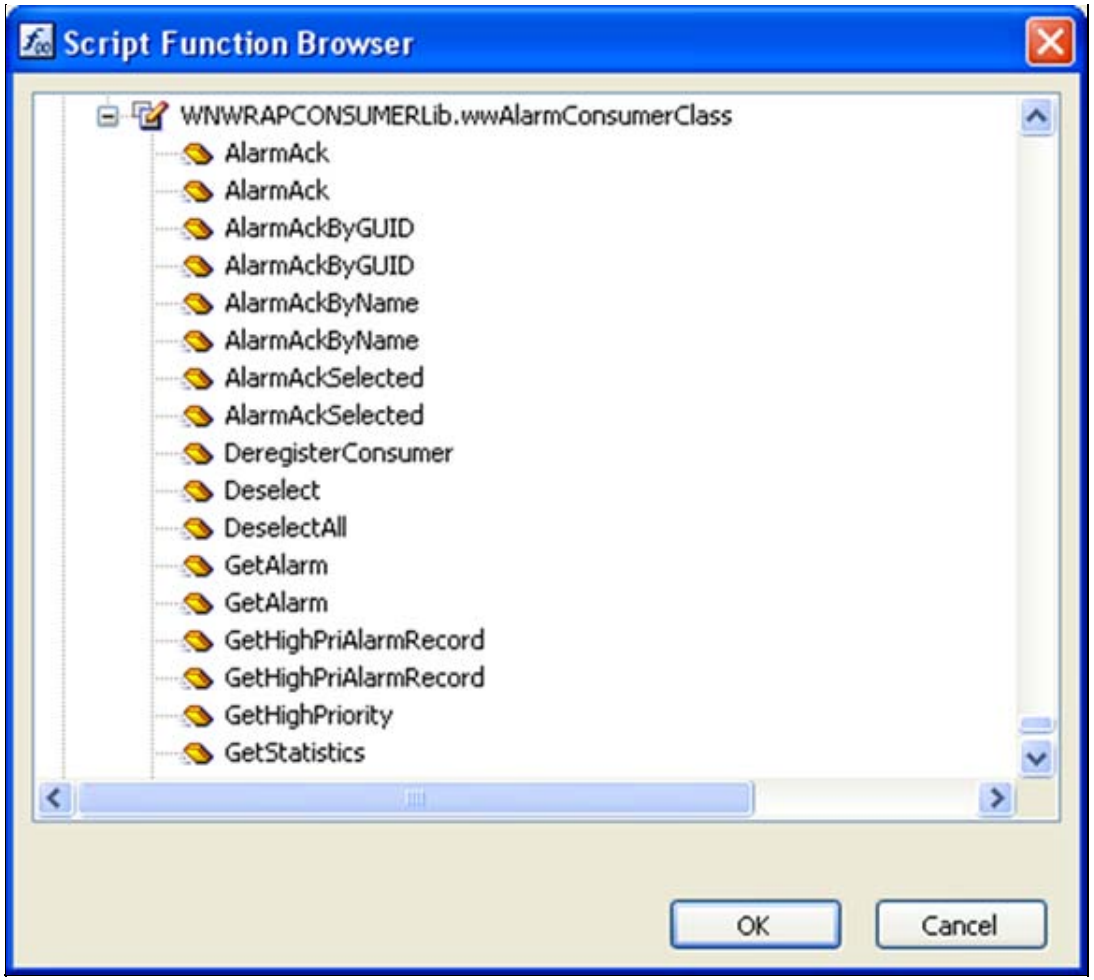

Figure 5: Script Function Browser

Creating the Scripts

Declare Script Variables

As per good programming practices, variable declarations are done in the Script editor's **Declarations** section.

```
Dim MyConsumer as WNWRAPCONSUMERLib.wwAlarmConsumerClass;
Dim currentXMLAlarms As Object;
Dim Result as Integer;

Dim almStr as String;
Dim xDoc as System.Xml.XmlDocument;
Dim node as System.Xml.XmlNodeList;
Dim leafnode as System.Xml.XmlNode;
```

FIGURE 6: DIMENSION DECLARATIONS

Script Example

```
Dim MyConsumer as WNWRAPCONSUMERLib.wwAlarmConsumerClass;
Dim currentXMLAlarms As Object;
Dim Result as Integer;

Dim almStr as String;
Dim xDoc as System.Xml.XmlDocument;
Dim node as System.Xml.XmlNodeList;
Dim leafnode as System.Xml.XmlNode;
```

WHERE

- **MyConsumer** – in the Quickscript is used for Alarm Consumer Methods.
- **currentXMLAlarms** – Alarm Records are returned in XML format.
- **Result** – An integer to hold the result of Initialize and Register Alarm Consumer Methods.
- **almStr** – Temporary storage for Alarm Records in XML format
- **xDoc**, **node** and **leafnode** variables are used to parse the XML Alarm records string to break it into a single alarm record and then further break into details within each alarm record like GUID, DATE, TIME, TAGNAME, TYPE, VALUE and STATE.

Create Your Startup Script

- **Execution Type: Startup**: Startup script is called when an object containing the script is loaded into memory, such as during deployment, platform, or engine start. Startup instantiates COM objects and .NET objects.

  Depending on load and other factors, assignments to object attributes from the Startup method can fail.

- Attributes that reside off-object are not available to the Startup method.

```
MyConsumer = new WNWRAPCONSUMERLib.wwAlarmConsumerClass;
xDoc = new System.Xml.XmlDocument;

Result = MyConsumer.InitializeConsumer("ConsumerApplication");
LogMessage(StringFromIntg(Result, 10));
Result = MyConsumer.RegisterConsumer(0, "testConsumer",
"ConsumerApplication", "1.1.1");
LogMessage(StringFromIntg(Result, 10));
LogMessage("---Instantiate, Initalize, Register AlarmConsumer,
SetXMLAlarmQuery---");

MyConsumer.SetXmlAlarmQuery("<QUERIES FROM_PRIORITY=""1""
TO_PRIORITY=""999""  ALARM_STATE=""All""
DISPLAY_MODE=""Summary""><QUERY><NODE>localhost</NODE><PROVIDER>Galaxy</PR
OVIDER><GROUP>Area_001</GROUP></QUERY></QUERIES>");

System.AppDomain.CurrentDomain.SetData("AlarmConsumerApp", MyConsumer);
```

FIGURE 7: STARTUP SCRIPT

Script Example

```
MyConsumer = new WNWRAPCONSUMERLib.wwAlarmConsumerClass;
xDoc = new System.Xml.XmlDocument;

Result = MyConsumer.InitializeConsumer("ConsumerApplication");
LogMessage(StringFromIntg(Result, 10));
Result = MyConsumer.RegisterConsumer(0, "testConsumer", "ConsumerApplication", "1.1.1");
LogMessage(StringFromIntg(Result, 10));
LogMessage("---Instantiate, Initalize, Register AlarmConsumer, SetXMLAlarmQuery---");

MyConsumer.SetXmlAlarmQuery("<QUERIES FROM_PRIORITY=""1"" TO_PRIORITY=""999""  ALARM_STATE=""All""
DISPLAY_MODE=""Summary""><QUERY><NODE>localhost</NODE><PROVIDER>Galaxy</PROVIDER><GROUP>Area_001</GROUP></QUERY></QUERIES>");

System.AppDomain.CurrentDomain.SetData("AlarmConsumerApp", MyConsumer);
```

This script does the following:

- Creates an instance of the Alarm consumer class **WNWRAPCONSUMERLib.wwAlarmConsumerClass** and **XmlDocument** class.
- Calls methods of the Alarm Consumer class to Initialize and Register the consumer: **InitializeConsumer** and **RegisterConsumer**. The **InitializeConsumer** method ensures that Alarm Manager has been started and Alarm system has been initialized.
  The **RegisterConsumer** registers with the Distributed Alarm System, with a Product Name of **testConsumer** and Application Name of**ConsumerApplication**.
- Calls **SetXmlAlarmquery** method Alarm Consumer class to set the Alarm Query. In this case Alarm Query is set to consume all alarms from the Galaxy under the **Area_001** group.
- The **System.AppDomain.CurrentDomain.SetData** method is used to share the MyConsumer Object connection with another script within the same or another Application Server object instance. In this case the **Myconsumer** Object is shared with script **AckAlarmByName** within the same object instance: **UD_AlmConsumer**.

Note: For more details on Alarm Toolkit Class Methods, refer to the **AlarmToolkit Users Guide on WDN**.

Create Execution Script Trigger

Create an Execution Script Trigger with the following:

Execution Type: Execute: This script is configured to trigger when AppEngine performs a scan, the Object is OnScan and when the boolean UDA **bConsumeAlms** is set to **TRUE**.

Figure 8: Execution Script Trigger

```
LogMessage("----------------------GetAlarms-----------------------------");
MyConsumer.GetXmlCurrentAlarms2(100, currentXMLAlarms);

almStr = currentXMLAlarms.ToString();
LogMessage(almStr);

'Load XML Alarms
xDoc.LoadXml(almStr);

node = xDoc.SelectNodes("/ALARM_RECORDS/ALARM");
LogMessage("----------------------------------------------------------------");

'node.InnerText property Get the concatenated values of the node and all
its child nodes
FOR EACH leafnode IN node
  LogMessage("Alarm Record.....................................");
  LogMessage(leafnode["GUID"].InnerText);
  LogMessage(leafnode["DATE"].InnerText);
  LogMessage(leafnode["TIME"].InnerText);
  LogMessage(leafnode["TAGNAME"].InnerText);
  LogMessage(leafnode["TYPE"].InnerText);
  LogMessage(leafnode["VALUE"].InnerText);
  LogMessage(leafnode["STATE"].InnerText);
NEXT;

me.bConsumeAlms = false;
```

FIGURE 9: SCRIPT

Script Example

```
LogMessage("----------------------GetAlarms-----------------------------");
MyConsumer.GetXmlCurrentAlarms2(100, currentXMLAlarms);

almStr = currentXMLAlarms.ToString();
LogMessage(almStr);

'Load XML Alarms
xDoc.LoadXml(almStr);

node = xDoc.SelectNodes("/ALARM_RECORDS/ALARM");
LogMessage("----------------------------------------------------------------");

'node.InnerText property Get the concatenated values of the node and all its child nodes
FOR EACH leafnode IN node
  LogMessage("Alarm Record.....................................");
  LogMessage(leafnode["GUID"].InnerText);
  LogMessage(leafnode["DATE"].InnerText);
  LogMessage(leafnode["TIME"].InnerText);
  LogMessage(leafnode["TAGNAME"].InnerText);
  LogMessage(leafnode["TYPE"].InnerText);
  LogMessage(leafnode["VALUE"].InnerText);
  LogMessage(leafnode["STATE"].InnerText);
NEXT;

me.bConsumeAlms = false;
```

- The boolean UDA **bConsumeAlms** is set to **TRUE** using the Object Viewer. This triggers the execution of the script. The **GetXmlCurrentAlarms2** method from the Alarm Consumer class is used to get all the alarm records in accordance with the Alarm Query set in an earlier section.
- The alarms in XML form are then copied in a temporary string called **almStr**.

- The classes **System.Xml.XmlDocument**, **System.Xml.XmlNodeList** and **System.Xml.XmlNode** are then used to separate out each individual element from the Alarm Records as shown above in the code snippet.

  After this script executes, the boolean UDA **bConsumeAlms** is set back to **FALSE** to ensure that Alarms can be requested on a demand basis and not continuously at every scan cycle. Also the LogMessage is used not only for logging alarm records but also used as debugging tool to monitor the progress of the script execution.

Create Execution Script Type: Shutdown

A Shutdown script is called when the object is about to removed from memory, usually as a result of the AppEngine stopping. Shutdown scripts are primarily used to destroy COM objects and .NET objects and to free memory.

```
LogMessage("---DeRegisterConsumer---");

MyConsumer.DeregisterConsumer();
```

Figure 10: Shutdown Script

Script Example

```
LogMessage("---DeRegisterConsumer---");
MyConsumer.DeregisterConsumer();
```

Acknowledging Alarms Using Application Server QuickScript

Create UDAs

1. In the **UD_AlmConsumer** object instance in ArchestrA IDE and select the UDAs tab.
2. Create following UDAs that will be used in the script:

- Booleans: **bAckAlarm** – used to enable the script for Acknowledging Alarm
- String: **strAlarmName** – Alarm Name to be acknowledged.

Create Script

Create a second script for acknowledging alarm by name, and call it **AckAlarmByName**.

Declare Script Variables

Declarations

```
Dim MyConsumer as WNWRAPCONSUMERLib.wwAlarmConsumerClass;
Dim Result as Integer;
```

Figure 11: Declarations

- **Execution Type: Execute** – This script is configured to trigger when AppEngine performs a scan, the Object is OnScan and when the boolean UDA **bAckAlarm** is set to **TRUE**.
- Execute Script:

```
MyConsumer = System.AppDomain.CurrentDomain.GetData("AlarmConsumerApp");

Result = MyConsumer.AlarmAckByName(me.strAlarmName, "\\localhost\Galaxy",
"Area_001", "ack from
ppk","oprNameisPPK",System.Environment.MachineName,System.Environment.UserD
omainName, System.Environment.UserName);

me.bAckAlarm = false;
```

Figure 12: Execute Script

Script Example

```
MyConsumer = System.AppDomain.CurrentDomain.GetData("AlarmConsumerApp");

Result = MyConsumer.AlarmAckByName(me.strAlarmName, "\\localhost\Galaxy", "Area_001", "ack from
ppk","oprNameisPPK",System.Environment.MachineName,System.Environment.UserDomainName, System.Environment.UserName);

me.bAckAlarm = false;
```

The **System.AppDomain.CurrentDomain.GetData** method gets the value stored in the current application domain for MyConsumer. The **strAlarmName** UDA is of type string and can be configured for a default value. For example: **UD_AlmConsumer.Analog_001.LoLo**. The string can be changed at runtime to acknowledge other alarms by name.

Testing Application Server QuickScript for Consuming and Acknowledging Alarms

Note: The InTouch Alarm Provider must be enabled to consume Alarms (Figure 13 below).

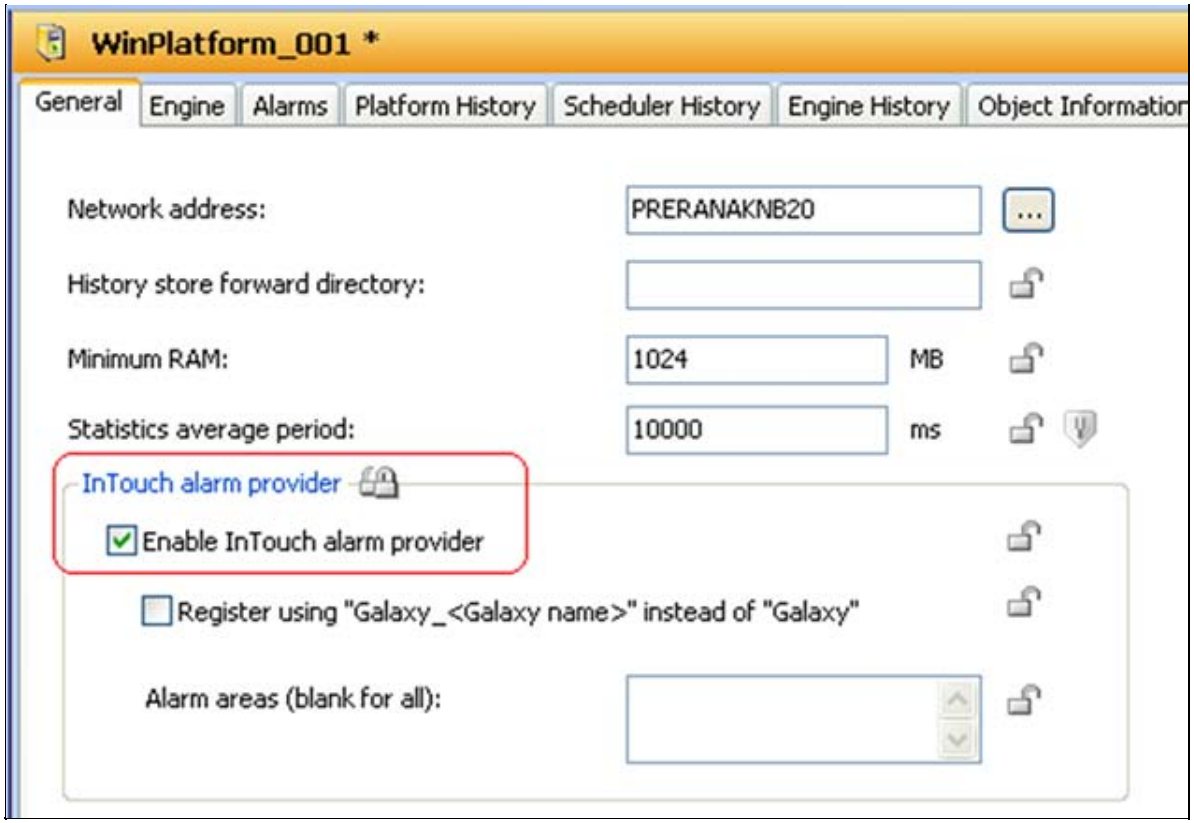

FIGURE 13: INTOUCH ALARM PROVIDER CONFIGURATION

1. Deploy **UD_AlmConsumer**.
2. Set up a Field Attribute for generating Alarms within the Area_001 in the Galaxy. For example - Generate Lo, Hi, HiHi alarms.
3. Start Object Viewer and set up a Watch Window with the following AttributeReferences (Figure 14 below):

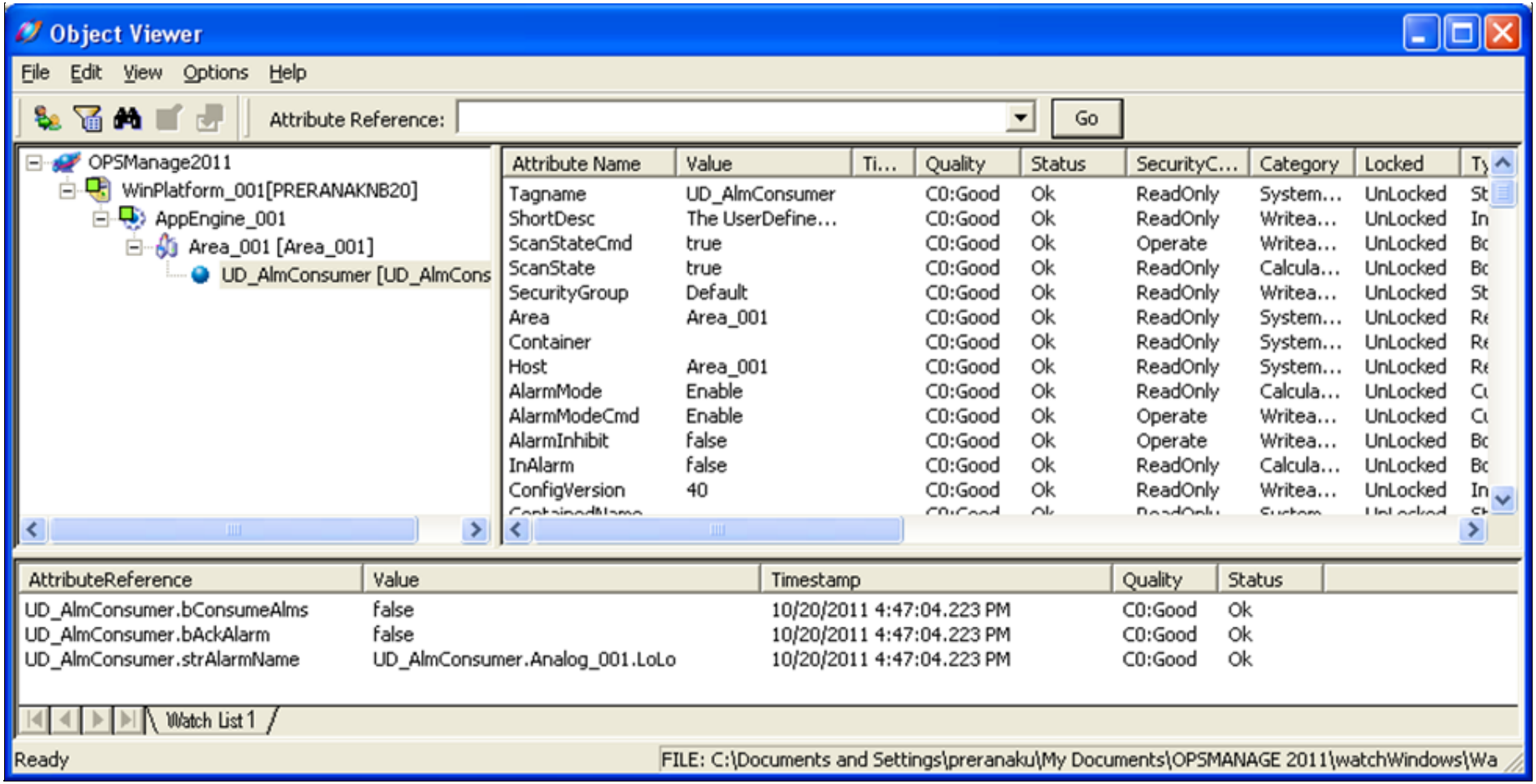

FIGURE 14: OBJECT VIEWER FOR UD_ALMCONSUMER OBJECT

4. Set **bConsumeAlms** to true. The **AlmCons** script will execute.
5. Look at the Log Viewer to see the logged Alarm Records (Figure 15 below).

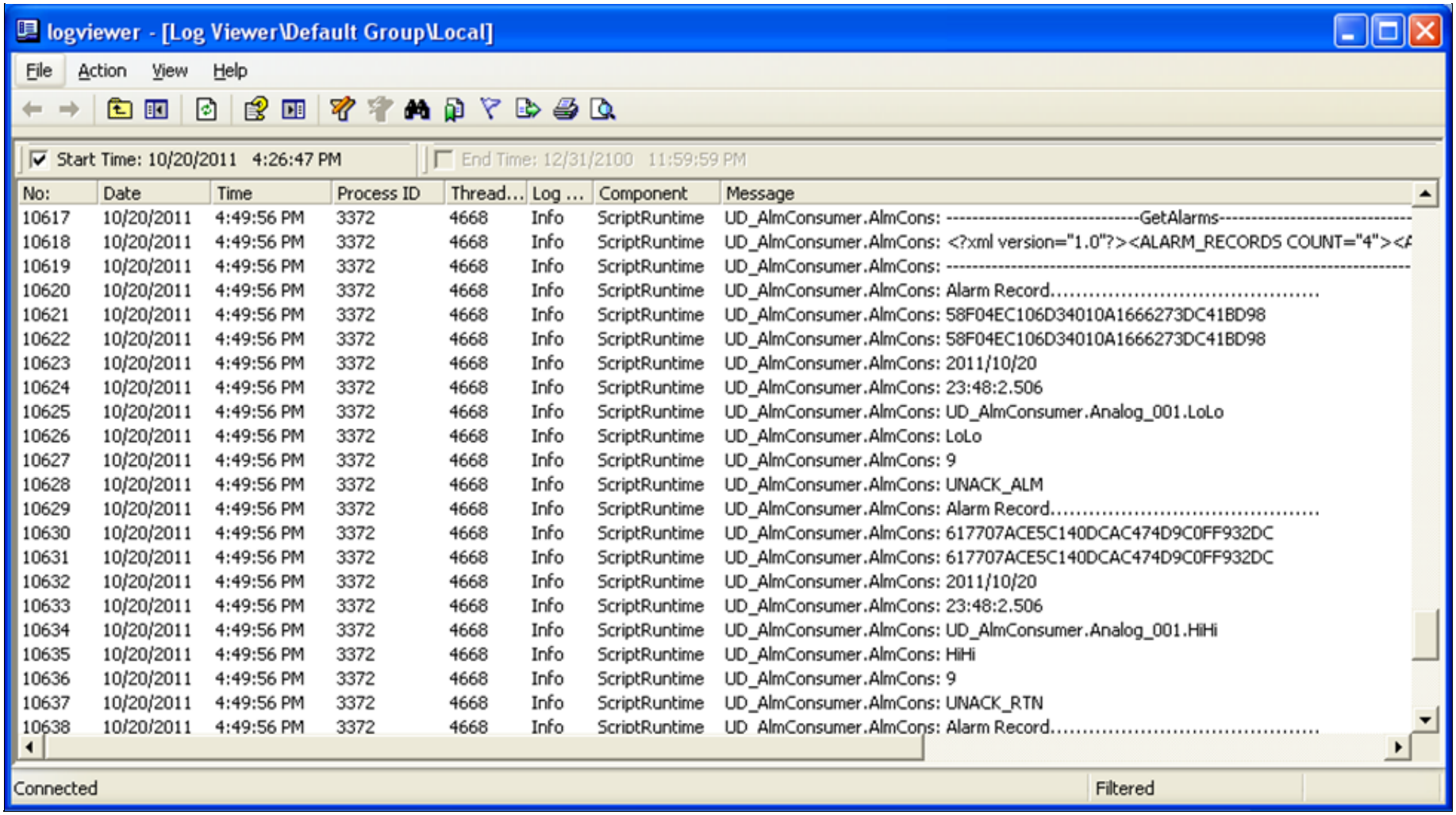

FIGURE 15: LOG VIEWER ALARM RECORDS

6. Set **bAckAlarm** to true. The **AckAlarmByName** script executes.
7. To monitor that the falarm for **UD_AlmConsumer.Analog_001.LoLo** was acknowledged, repeat steps 4 and 5.

   Notice in the logger that Alarm record for **UD_AlmConsumer.Analog_001.LoLo** alarm has status of **ACK_ALM**.

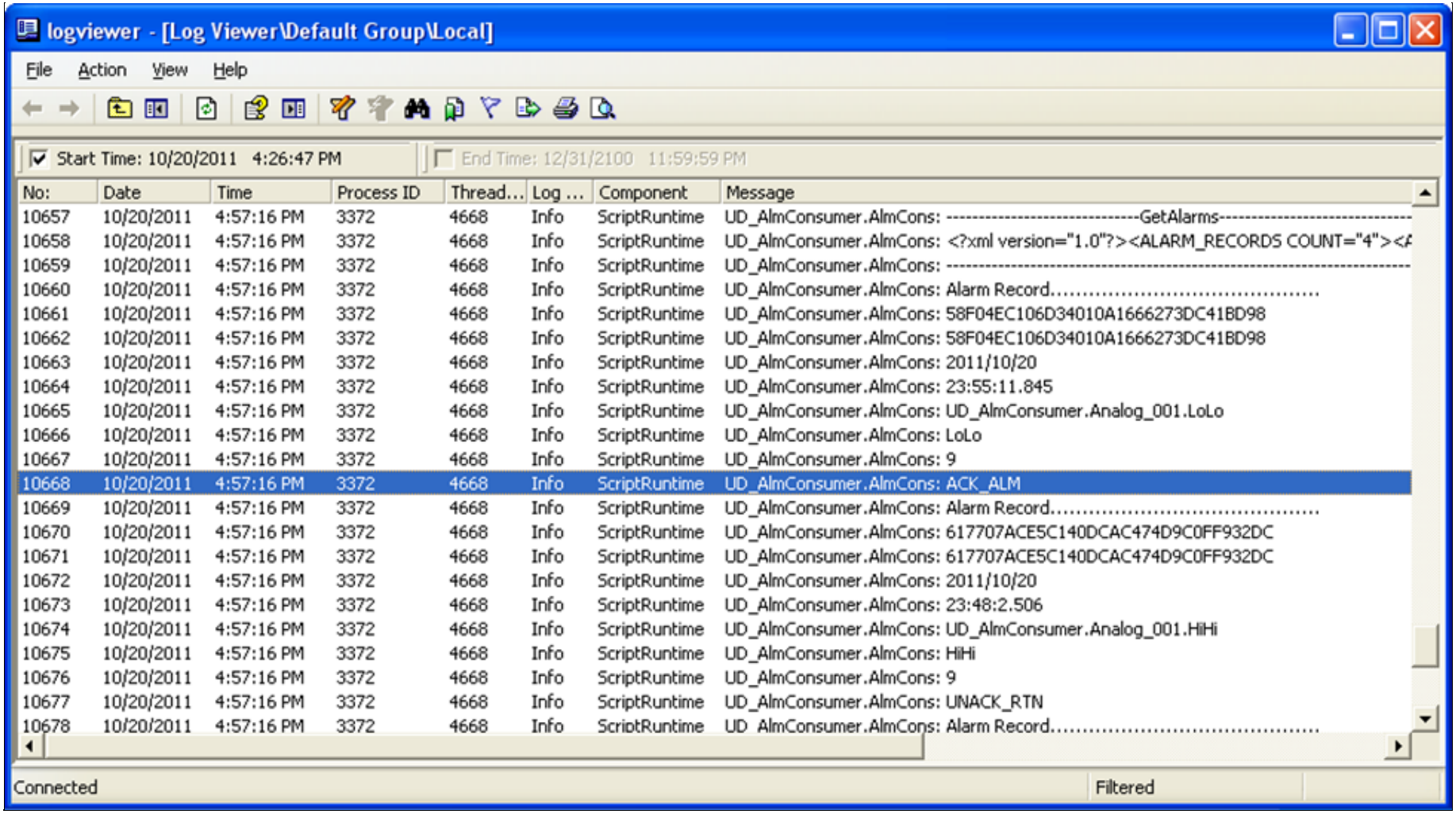

FIGURE 16: ACK_ALM STATUS

P. Kulkarni

Tech Notes are published occasionally by Wonderware Technical Support. Publisher: Invensys Systems, Inc., 26561 Rancho Parkway South, Lake Forest, CA 92630. There is also technical information on our software products at **Wonderware Technical Support.**

For technical support questions, send an e-mail to **wwsupport@invensys.com**.

Back to top