EnterpriseAlert® 2017
HTTP/SOAP API & Connector SDK

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

1 PREFACE

Enterprise Alert® comes with a powerful 2-way HTTP/SOAP API to enable the design of SOA-oriented and SOA-compliant applications and services (SOA - Service Orientated Architecture).

This API enables you to integrate your specific application requirements with the powerful messaging service provided by Enterprise Alert®. The API provides direct access to the events and processes as they occur within the messaging service.

The messaging service of Enterprise Alert® is event based, which means that notifications or alerts that are sent out are initially triggered by incoming messages or events. These messages can come in the form of a SMS, MMS, e-mail, instant Message or voice message. Events on the other hand can come from any event source or event provider.

Enterprise Alert® already comes with a number of pre-defined event sources such as Microsoft Operations Manager or the Serial Connector. But by no means are the types of events that are available restricted to these event sources. The architecture of Enterprise Alert® enables 3rd party event providers to register their applications as event sources in Enterprise Alert®. This architecture provides a complete solution for the creation of events and the monitoring of their statuses.

After the event provider has been registered, the event provider will show up in the Web portal of Enterprise Alert®, and can then be used in the Event Filter just like any other event source to send out notifications or create alerts. Besides this, the event provider can register its own custom parameters, which can then be used to filter out only the events that are of interest for sending out messages.

This API supports the following features:

- Self registration of 3rd party event providers
- Ability to provide event parameter information for use in the Web portal
- Raising events
- Request of event and alert statuses
- Active forwarding of alert status change events to 3rd party event providers.
- HTTP GET, HTTP POST as well as SOAP/Web service support

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

2 HTTP/SOAP API INTRODUCTION

The Enterprise Alert® HTTP/SOAP API handles events from 3rd party event providers and status information requests. Event providers can register their event types and corresponding parameters. Various methods/functions are available in the HTTP/SOAP API:

Method	Description
About	About method
RegisterEventProvider	Registers an event provider in Enterprise Alert® with meta information about the events and the parameters
UnregisterEventProvider	Unregisters an event provider in Enterprise Alert®
RaiseEvent / RaiseEventAsync	Raises an event
GetEventStatus	Requests the status of a previously raised event
ResetEvent	Resets an event, which was raised
GetAlertStatus	Request the status of an alert
CloseAlert	Closes an alert

Chapters 2-5 of section B describe the architecture, functionality and how to use the HTTP/SOAP API for registering event providers and raising events. A .NET sample application that acts as an event provider is given in chapter 6. Chapters 7-8 describe the automated pushing of ticket status change events to registered event providers through the client component of the HTTP/SOAP API. How to implement a Web service/Web application for handling such events is described in the chapter as well.

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

3 ARCHITECTURE

Web requests for registering event providers, raising events and requesting event statuses are handled by the server component of the HTTP/SOAP API.

The client component of the HTTP/SOAP API forwards ticket status change events to registered 3rd party event providers.

The server component uses Microsoft Internet Information Services (IIS) to provide the under lying HTTP server functionality. The following request formats are currently implemented:

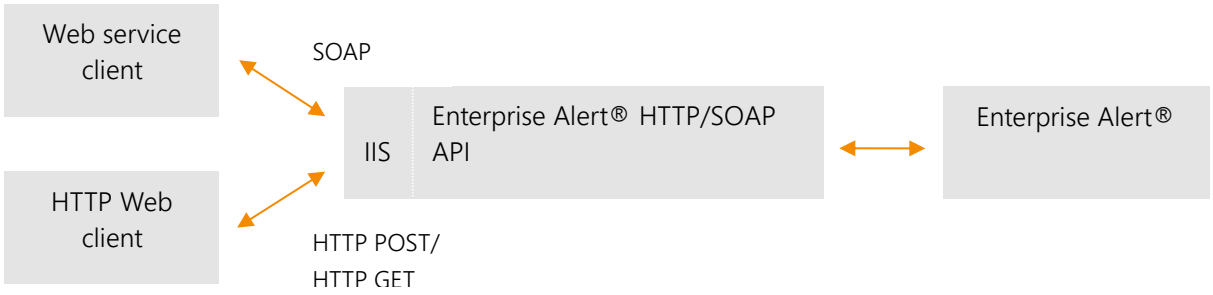
- SOAP/ Web service
- HTTP GET
- HTTP POST

The client component supports the following HTTP formats:

- SOAP/ Web service
- HTTP POST

The figure below shows the architecture of the Enterprise Alert® HTTP/SOAP API. The interface is implemented as a .NET 3.5 Web application and is hosted in Microsoft Internet Information Services (IIS).

The API contains a Web service for handling requests from Web service clients via SOAP and a Web page for handling HTTP client POST and GET requests. On the other side, the HTTP/SOAP API communicates with the Enterprise Alert® server via Microsoft Message Queuing (MSMQ) using the common Enterprise Alert® protocol.



The HTTP/SOAP API provides several functions for communicating with the Enterprise Alert® server. These function calls are translated into XML messages and then sent to the Enterprise Alert® server or vice versa.

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

4 SYSTEM REQUIREMENTS

The Enterprise Alert® HTTP/SOAP API requires Microsoft Internet Information Services (IIS) as a hosting environment. IIS is part of the standard Windows components and can be installed using the *Windows Component Wizard*. To install, open *Add or Remove Programs* in the *Control Panel*. Click on *Windows Components* and select *Internet Information Services* in the opened *Windows Component Wizard*. Next, select OK to start the installation of the IIS.

The HTTP/SOAP API is based on the .NET Framework 3.5. Therefore the .NET Framework 3.5 must be installed and registered in Internet Information Services (IIS) prior to installation. The setup program of the .NET Framework 3.5 can be downloaded from the following link on the Microsoft Web site:

HTTP://www.microsoft.com/downloads/details.aspx?familyid=AB99342F-5D1A-413D-8319-81DA479AB0D7

5 WSDL DOCUMENT

For accessing the API's Web service using SOAP you can use a WSDL document that provides a description of the interface functions. This document can be found at the following URL:

HTTP://localhost/EAWebService/EventProviderAPI.asmx?WSDL

localhost should be replaced by the name of the machine where the Enterprise Alert® HTTP/SOAP API is installed.

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

6 FUNCTION INTERFACE

This chapter describes the functions of the implemented interface.

6.1 About

This function provides copyright information about the Enterprise Alert® HTTP/SOAP API.

Return value

SOAP / .NET	0 if the function succeeded, otherwise an error code
HTTP GET / HTTP POST	The HTTP request returns a page with the following text: [Copyright information about this Enterprise Alert® interface.]

Samples

HTTP GET:

HTTP://localhost/EAWebService/EventProviderAPI.aspx?Handler=About

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

6.2 RegisterEventProvider

This function registers a 3rd party event provider in Enterprise Alert®.

Parameters

Parameter	Description	Data type
Username	Username of the user registering the event provider	String (max length: 100)
Password	The password for the user registering the event provider	String (max length: 100) or md5 hashed
ProviderName	The name for the event provider to register	String (max length: 100)
EventParameters	Names of the event parameters separated by semicolon	String (max length: 255 per parameter name)
ResponseURI	URI of a Web service/Web application where alert status change events should be sent.	String (max length: 255)

Return value

SOAP / .NET	The internal provider ID if the function succeeded, otherwise an HTTP SOAP fault containing the error details.
HTTP GET / HTTP POST	The HTTP request returns a page with the following text: In case of success: ProviderID: [The internal provider ID of the registered event provider] In case of failure: Error: [An error number. -1 means an unspecified error.] ErrorText: [A description for the error that occurred]

Samples

HTTP GET:

HTTP://localhost/EAWebService/EventProviderAPI.aspx?Handler=RegisterEventProvider&Username=Administrtor&Password=&ProviderName=MyProvider&EventParameters=param1;param2&ResponseURI=HTTP://localhost/EAEventHandler.aspx

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

6.3 UnregisterEventProvider

This function unregisters an existing 3rd party event provider in Enterprise Alert®.

Parameters

Parameter	Description	Data type
Username	Username of the user unregistering the event provider	String (max length: 100)
Password	The password for the user unregistering the event provider	String (max length: 100) or md5 hashed
ProviderName	The name of the event provider to unregister	String (max length: 255)

Return value

SOAP / .NET	Nothing if the function succeeded, otherwise an HTTP SOAP fault containing the error details.
HTTP GET / HTTP POST	The HTTP request returns a page with the following text: In case of success: OK In case of failure: Error: [An error number. -1 means an unspecified error.] ErrorText: [A description for the error that occurred]

Samples

HTTP GET:

HTTP://localhost/EAWebService/EventProviderAPI.aspx?Handler=UnregisterEventProvider&Username=Administrator&Password=&ProviderName=MyProvider

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

6.4 RaiseEvent / RaiseEventAsync

These functions raise an event in Enterprise Alert®. RaiseEvent is a synchronous function and returns the generated EventID. RaiseEventAsync raises the event asynchronous in Enterprise Alert® and does not return the EventID. Therefore the ExternalID must be used for Event Status requests or resetting event.

Parameters

Parameters	Description	Data type
Username	The username of the user raising the event	String (max length: 100)
Password	The password for the user raising the event	String (max length: 100) or md5 hashed
ProviderName	The name of the event provider	String (max length: 255)
EventParameters	The parameters of the event in key-value format	String (max length: 255 per parameter name, 255 per parameter value)
AttachmentURIs	URIs to binary attachments (jpg, gif, ...) which should be assigned to the event separated by semicolons.	String (max length: 255)
ExternalID	This field can be used for specifying an external ID for the event.	String (max length: 255)

Return value

SOAP / .NET	The generated Event ID (Integer) if the function succeeded, otherwise an HTTP SOAP fault containing the error details.
HTTP GET / HTTP POST	The HTTP request returns a page with the following text: In case of success: EventID: [The ID of the raised event for later use] In case of failure: Error: [An error number. -1 means an unspecified error.] ErrorText: [A description for the error that occurred]

Attachments

You can send attachments either as HTTP links (via POST or GET) by setting these links in the parameter ***AttachmentURIs*** or you can use file input HTML tags e.g. ***<INPUT TYPE="file" .. >***(as many as you want) if you want to upload files. In the former case, you must ensure that the links are accessible by the Enterprise Alert® server via the Internet. In the latter case, you must use HTTP POST, whereby you can use any name for the ***<INPUT TYPE="file"...>*** tags, as long as the names used are unique.

If you use the Web service interface, you have to put the attachment contents in *Attachment* data objects as defined in the Web service description. Binary content should be encoded in base64. These *Attachment* objects must be provided as an array to the parameter *Attachments* of the method *RaiseEvent()*.

Samples

HTTP GET:

HTTP://localhost/EAWebService/EventProviderAPI.aspx?Handler=RaiseEvent&Username=Administrator&Password=&ProviderName=MyProvider&param1=a&param2=b&AttachmentURIs=HTTP://www.derdack.com/files/logo_derdack.png

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

6.5 GetEventStatus

This function requests the status of an event in Enterprise Alert®.

Parameters

Parameter	Description	Data type
Username	Username of the user requesting the event status	String (max length: 100)
Password	The password for the user requesting the event status	String (max length: 100) or md5 hashed
ProviderName	The event provider name which has raised the event	String (max length: 255)
EventID	The unique ID of the event, provided by Enterprise Alert at raising the event	Integer
ExternalEventID	The external Event ID provided by the event provider at raising the event. (Optional instead of EventID)	String (max length: 255)

Return value

SOAP / .NET	A Status structure if the function succeeded, otherwise an HTTP SOAP fault containing the error details.
HTTP GET / HTTP POST	The HTTP request returns a page with the following text: In case of success: EventID: [The unique ID of the event] AppliedPolicies: [Number of policies triggered by the event] Duplicates: [Number of policies where duplicate detection have applied by the event] AlertsDelayed: [Number of delayed alerts created by the event] AlertsCreated: [Number of alerts created by the event] AlertIDs: [Semicolon separated list of unique IDs of the created alerts] In case of failure: Error: [An error number]: - 1: The event was not found - -1: Unspecified error. ErrorText: [A description for the error that occurred]

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

Samples

HTTP GET:

HTTP://localhost/EAWebService/EventProviderAPI.aspx?Handler=GetEventStatus&Username=Administrator&Password=&ProviderName=MyProvider&EventID=18421

6.6 ResetEvent

This function resets an event in Enterprise Alert®, which was previously raised.
If an alert was opened by the specified previous event, the alert will be closed.
If a delayed alert was created by the event, the alert is never being started.
If no alert was created by the event, the function will return without any modifications.

Parameters

Parameter	Description	Data type
Username	Username of the user submitting the reset event	String (max length: 100)
Password	The password for the user submitting the reset event	String (max length: 100) or md5 hashed
ProviderName	The event provider name which has raised the event	String (max length: 255)
EventID	The unique ID of the event, provided by Enterprise Alert on raising the previous event	Integer
ExternalEventID	The external Event ID provided by the event provider on raising the previous event. This is optional and can be used instead of EventID.	String (max length: 255)
Description	Reason for resetting the event. This text will be provided in notification messages to managers, alert owners etc.	String (max length: infinite)

Return value

SOAP / .NET	Nothing if the function succeeded, otherwise an HTTP SOAP fault containing the error details.
HTTP GET / HTTP POST	The HTTP request returns a page with the following text: In case of success: OK In case of failure: Error: [An error number. -1 means an unspecified error.] ErrorText: [A description for the error that occurred]

Samples

HTTP GET:

HTTP://localhost/EAWebService/EventProviderAPI.aspx?Handler=ResetEvent&Username=Administrator&Password=&ProviderName=MyProvider&EventID=42

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

6.7 GetAlertStatus

This function requests the status of an alert in Enterprise Alert®.

Parameters

Parameter	Description	Data type
Username	Username of the user requesting the alert status	String (max length: 100)
Password	The password for the user requesting the alert status	String (max length: 100) or md5 hashed
ProviderName	The event provider name which has raised the event	String (max length: 255)
AlertID	The unique ID of the alert	Integer

Return value

SOAP / .NET	A Status structure if the function succeeded, otherwise an HTTP SOAP fault containing the error details.
HTTP GET / HTTP POST	The HTTP request returns a page with the following text: In case of success: AlertID: [The unique ID of the alert] Status: [The status of the alert as constant text.] - Open - Acknowledged - Closed - Failed - Error - Declined - NoReply - Canceled - Forwarded StatusCode: [The status code as number assigned to the alert status] - 1: Open - 2: Acknowledged - 3: Closed - 4: Failed

- 5: Error
- 6: Declined
- 7: NoReply
- 8: Canceled
- 10: Forwarded

Reason: [Description of the last alert status change]

StatusChangeUser: [Username of the user which has changed the alert status]

In case of failure:

Error: [An error number]

- 1: The alert was not found
- -1: Unspecified error.

ErrorText: [A description for the error that occurred]

Samples

HTTP GET:

HTTP://localhost/EAWebService/EventProviderAPI.aspx?Handler=GetAlertStatus&Username=Administrator&Password=&EventID=42

6.8 CloseAlert

This function closes an alert in Enterprise Alert®.

Parameters

Parameter	Description	Data type
Username	Username of the user closing the alert	String (max length: 100)
Password	The password for the user closing the alert	String (max length: 100) or md5 hashed
ProviderName	The name of the event provider closing the alert	String (max length: 255)
AlertID	The unique ID of the alert	Integer
Description	Reason about closing the alert. This text will be provided in notification messages to managers, alert owners etc.	String (max length: infinite)

Return value

SOAP / .NET	None if the function succeeded, otherwise an HTTP SOAP fault containing the error details.
HTTP GET / HTTP POST	The HTTP request returns a page with the following text: In case of success: OK In case of failure: Error: [An error number. -1 means an unspecified error.] ErrorText: [A description for the error that occurred]

Samples

HTTPGET:

HTTP://localhost/EAWebService/EventProviderAPI.aspx?Handler=CloseAlert&Username=Administrator&Password=&ProviderName=MyProvider&AlertID=2&Description=ProblemResolved

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

6.9 SubmitEventUpdate

Similar to the *ResetEvent* function, this function resets an event in Enterprise Alert®, which was previously raised.
If an alert was opened by the previous event, the alert will be closed.
If a delayed alert was created by the previous event, the alert will never be started.
If no alert was created by the event, the function will return without any modifications.

The difference between *SubmitEventUpdate* and *ResetEvent* is that *SubmitEventUpdate* allows to specify event parameters, whereas *ResetEvent* does not. In addition, the reset functionality specified above is optional and is set via the *ResetEvents* parameter. If you do not wish to use the reset functionality, an update event will simply be created with the event parameters you provide. The update event can be distinguished from the new event via the "Is New Event" and "Is Update Event" parameters, which are automatically added depending on which function is called.

Another difference between the two functions is that *SubmitEventUpdate* may trigger an Alert Policy, while *ResetEvent* will not.

Parameters

Parameter	Description	Data type
Username	Username of the user submitting the event update	String (max length: 100)
Password	The password for the user submitting the event update	String (max length: 100) or md5 hashed
ProviderName	The name of the event provider for the update event	String (max length: 255)
EventID	The unique ID of the previous event, provided by Enterprise Alert on raising the event. This is optional when ResetEvents is not set.	Integer
ExternalID	The external Event ID provided by the event provider on raising the previous event. This is optional and can be used instead of EventID or when ResetEvents is not set.	
Description	Reason about closing the alert. This text will be provided in notification messages to managers, alert owners etc.	String (max length: infinite)
ResetEvents	Whether or not the previous event specified by the EventID or ExternalID parameters should be reset and also whether alerts opened by the previous event should be closed.	
EventParameters	The parameters of the event in key-value format.	

Return value

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

SOAP / .NET	None if the function succeeded, otherwise an HTTP SOAP fault containing the error details.
HTTP GET / HTTP POST	The HTTP request returns a page with the following text: In case of success: OK In case of failure: Error: [An error number. -1 means an unspecified error.] ErrorText: [A description for the error that occurred]

Samples

HTTPGET:

HTTP://localhost/EAWebService/EventProviderAPI.aspx?Handler=SubmitEventUpdate&Username=Administrator&Password=&ProviderName=MyProvider&EventID=2&Description=ProblemResolved&ResetEvents=true&ParamOne=Machine1&ParamTwo=ConnectivityUp

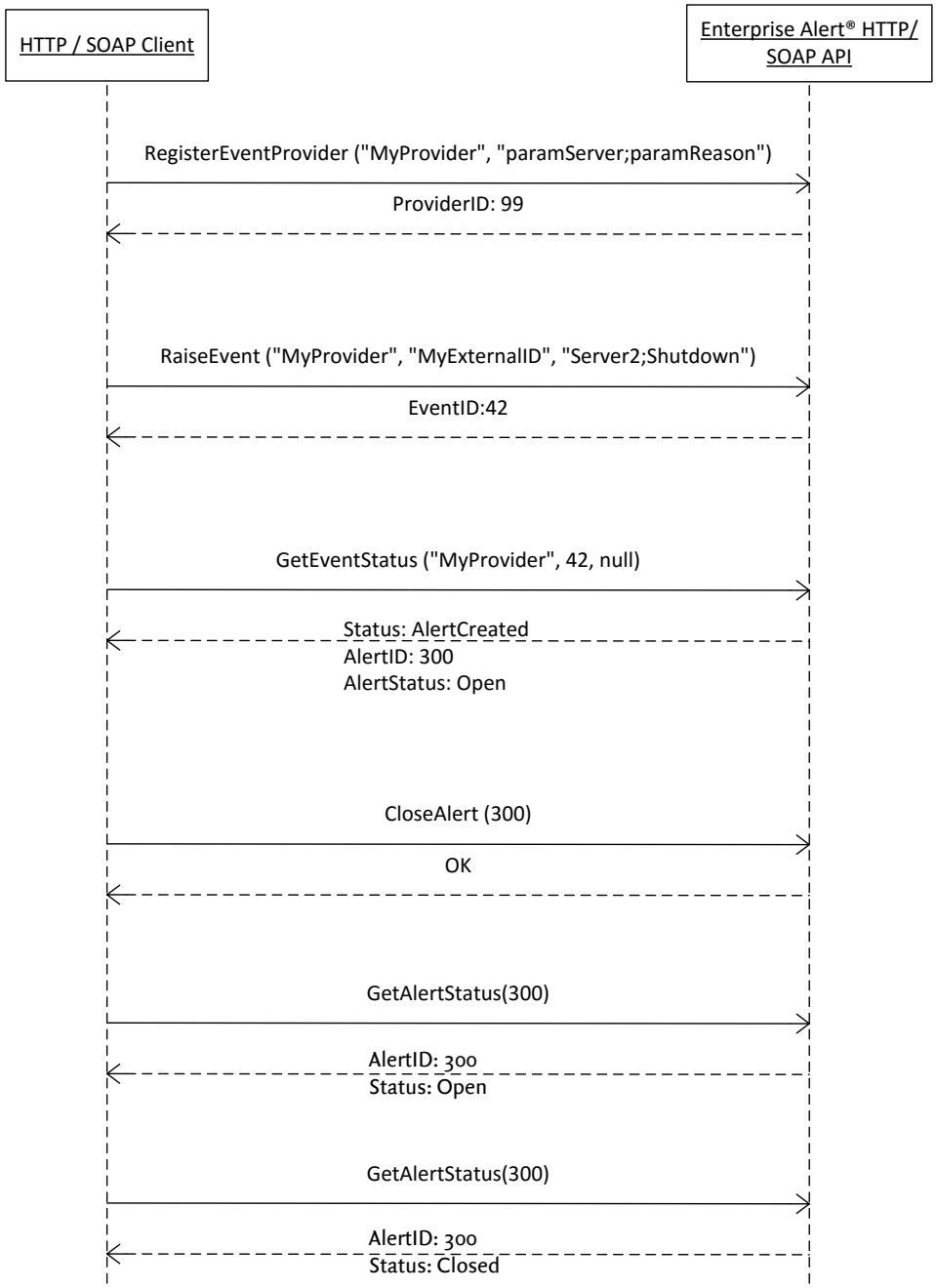
7 REQUIRED PERMISSIONS

The user account used for accessing the SOAP API can be either a custom Enterprise Alert® account or a Windows account which has been synchronized from Active Directory. The account needs to either be assigned the Administrators role or needs to be assigned to a user role that has the "Connect to Web Service API" permission. The latter is only supported if your license allows you to customized user roles and permissions.

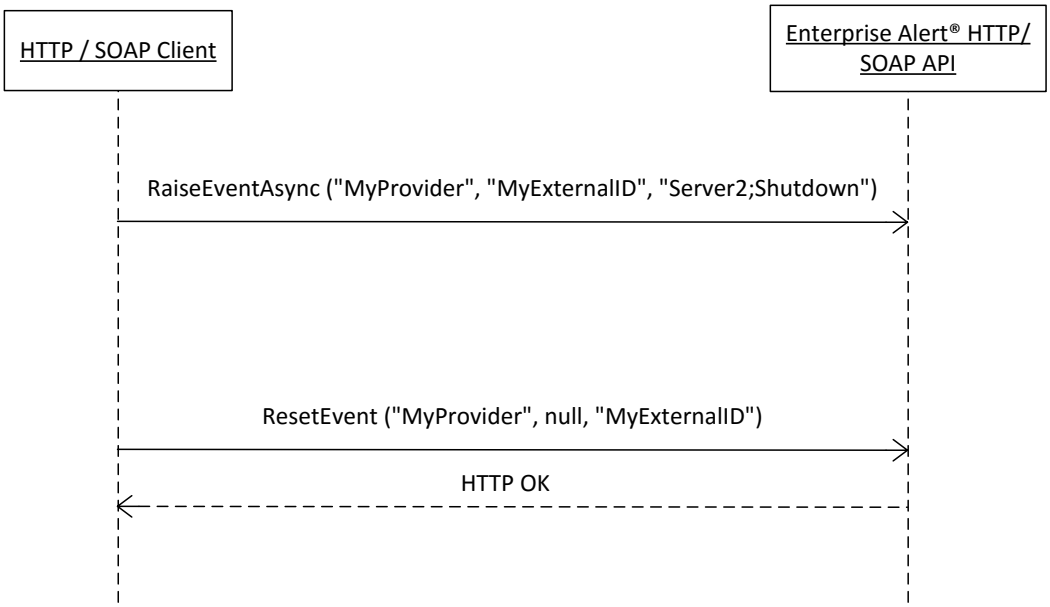
PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

8 SCENARIOS

8.1 Register an event provider, raise an event, get the status and close the according alert



8.2 Raise an event asynchronous and reset the event by external event ID



If an alert was already been opened by the event, the alert will be closed.

If a delayed alert was created by the event, the alert will not been started after the delay.

If no alert was created by the event, nothing will be done in Enterprise Alert.

9 .NET SAMPLE APPLICATION

Derdack provides a sample Windows form application that uses the HTTP/SOAP API. The sample is created in Microsoft Visual Studio 2008 and written in C#.

By default the .NET sample project can be found at *C:|Program Files/ Enterprise Alert|Samples|HTTP SOAP API|EAWebServiceClient*.

Below you find the main steps required to create the sample application.

9.1 Step 1: Creating the Project in Visual Studio

Open Visual Studio 2008 and select *New Project*. In the *New Project* dialog box, select *Visual C# Projects* and *Windows Forms* application.

9.2 Step 2: Designing the Main form

After the project has been created you should see a blank Windows form where all the Windows controls can be placed. For this sample, we need the following controls:

- *TextBoxUsername*: A text box for the username.
- *TextBoxPassword*: A text box for the password
- *TextBoxEventProviderName*: A text box for the name of the event provider
- *TextBoxEventParameters*: A text box for the event parameters
- *ButtonRegister*: A button for registering the event provider.
- *ButtonUnregister*: A button for unregistering the event provider.
- *ButtonRaiseEvent*: A button for raising an event.
- *LabelResult*: A text label for showing the result of each operation performed.

9.3 Step 3: Implementation

After having added all the Windows forms controls, we need to add a reference to the Enterprise Alert® Web service. To do so, please follow the steps below:

a. Right click the main project item in the Solution Explorer.

b. Select the menu item *Add Service Reference ….*

c. In the appearing *Add Service Reference* dialog box, enter the URL of the Enterprise Alert® SOAP API Web service description e.g. HTTP://localhost/EAWebService/EventProviderAPI.asmx?WSDL. After pressing Enter, you will see a description of all the functions that are available in the Web service.

d. Define a namespace e.g. *EnterpriseAlert* and click *Add Reference* in order to add the Web reference to your project.

e. Next, we will fill our application with some code. Simply double click on the *Register* button in the Windows form (*Form1.cs*). The event handler method *ButtonRegister_Click* will automatically be created. You can fill this method with the following code, which registers the event provider in Enterprise Alert®:

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

```
private void ButtonRegister_Click(object sender, EventArgs e)
{
      EventProviderAPIWSSOAPClient eventProviderAPI =
                    new EventProviderAPIWSSOAPClient();

      eventProviderAPI.Endpoint.Address =
             new System.ServiceModel.EndpointAddress(TextBoxUri.Text);

      // create the event parameter definitions
      //
      string[] astrParameterDefs = TextBoxEventParameter.Text.Split(
             new string[] { Environment.NewLine },
             StringSplitOptions.RemoveEmptyEntries
             );

      foreach (string strParam in astrParameterDefs)             {
          EventParameterDefinition parameterDef =
              new EventParameterDefinition()
              {
                     Name = strParam,
                     DisplayName = strParam,
                     Description = ""
              };

          listParameterDefs.Add(parameterDef);
      }

      try
      {
          int nResult = eventProviderAPI.RegisterEventProvider(
              TextBoxUsername.Text,
              TextBoxPassword.Text,
              TextBoxProviderName.Text,
              "",
              listParameterDefs.ToArray(),
              null);

          MessageBox.Show(string.Format("Event Provider ID={0}", nResult));
      }
      catch (Exception ex)
      {
          MessageBox.Show(string.Format("Error: {0}", ex.Message));
      }
}
```

f. To raise an event in Enterprise Alert®, we have to implement the click event handler for the ***RaiseEvent*** button. To do this, double click the ***RaiseEvent*** button to generate the method ***ButtonRaiseEvent_Click*** and fill the body with the following code:

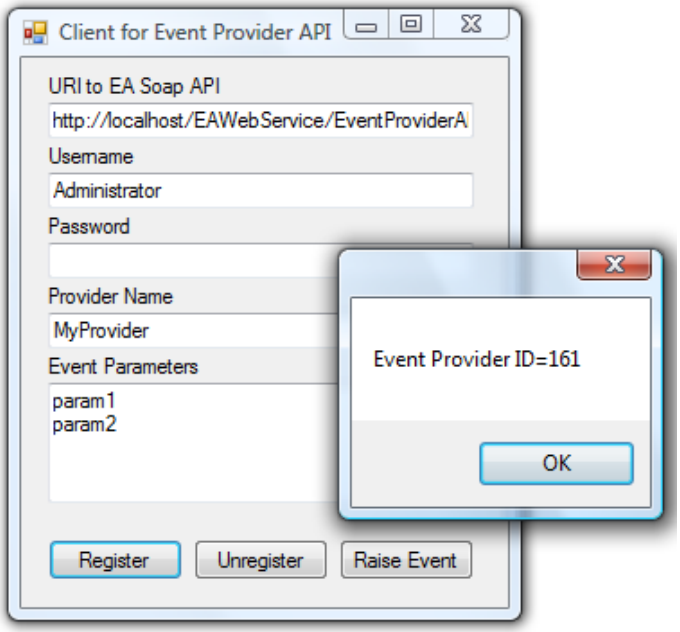
```
private void ButtonRaiseEvent_Click(object sender, EventArgs e)
{
      EventProviderAPIWSSOAPClient eventProviderAPI =
             new EventProviderAPIWSSOAPClient();

      eventProviderAPI.Endpoint.Address =
             new System.ServiceModel.EndpointAddress(TextBoxUri.Text);

      // create the event parameter definitions
      //
      List<EventParameter> listParameters = new List<EventParameter>();
```

```
        string[] astrParameter = TextBoxEventParameter.Text.Split(
                new string[] { Environment.NewLine },
                StringSplitOptions.RemoveEmptyEntries);

        foreach (string strParam in astrParameter)
        {
            // split into key and value
            //
            string[] strKeyValue = strParam.Split(new char[] { ';'}, 2);

            if (strKeyValue.Length == 2)
            {
                EventParameter parameter = new EventParameter()
                {
                    Name = strKeyValue[0],
                    Value =  strKeyValue[1]
                };
                listParameters.Add(parameter);
            }


        }


        try
        {
            int nResult = eventProviderAPI.RaiseEvent(
                TextBoxUsername.Text,
                TextBoxPassword.Text,
                TextBoxProviderName.Text,
                listParameters.ToArray(),
                null);

            MessageBox.Show(string.Format("Event ID={0}", nResult));
        }
        catch (Exception ex)
        {
            MessageBox.Show(string.Format("Error: {0}", ex.Message));
        }
    }
```

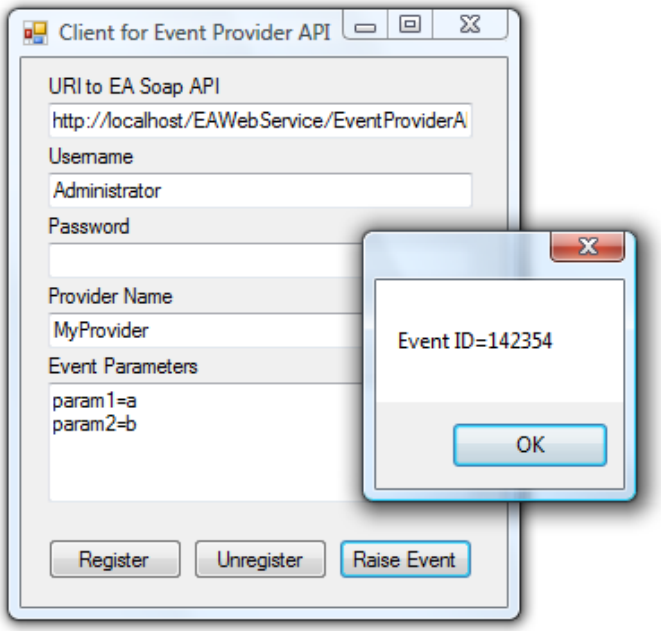
9.4 Step 4: Testing

We are now ready to test the client application.

a. Press F5 in Visual Studio to compile and start the application. The main form will be opened. Enter the username, password, provider name and event parameter names (one name per row). Then select ***Register***. If the registration is successful, the resulting ***Provider ID*** is displayed in the message box:



b. For raising an event, enter values for the event parameters by appending an '=' and the according value for each parameter name. Then select ***Raise Event***. If the event is successfully raised, the resulting Event ID is displayed in the message box:



10 ALERT STATUS UPDATE EVENTS FROM ENTERPRISE ALERT®

10.1 General

The client part of the Enterprise Alert® HTTP/SOAP API is able to forward alert events e.g. alert status change events to 3rd party Web applications and services.

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

10.2 Functionality

When an event provider registers itself by calling the HTTP/SOAP API method *RegisterEventProvider*, it can specify the address for a Web service or Web application that will receive alert status change events related to alerts created by the event provider.

Enterprise Alert® will send these event messages via HTTP POST or SOAP. This operation depends on the address provided when registering the event provider. If the provider was registered via HTTP POST or HTTP GET, the alert status change events will be forwarded via HTTP POST. If the provider was registered by calling the Web service via SOAP, the alert status change events will be forwarded via SOAP as well.

In order to handle alert status change events, a Web service or Web application must be implemented for the event provider. The format of the SOAP/POST messages is described in the following sections.

10.2.1 Handling of alert status change events in Web applications received via HTTP POST

Alert status change events via HTTP POST will be sent by Enterprise Alert® as a *multipart/form-data*-POST request with the following parameters. The Web application must be able to filter the parameters from the HTTP POST Web request:

Parameters

Parameter	Description
EventID	ID of the alert event in Enterprise Alert®
EventName	Name of the alert event. The value is always *OnTicketStatus*.
EventType	Type of the alert event. The value is always *TicketProcessing*.
Options	Options of the alert event. This parameter is reserved for later use and the value is always zero.
StringRef	A reference field. This parameter is reserved for later use and the value is always empty.
Timestamp	Timestamp of the event in format *YYYY-MM-DD hh:mm:ss*
TP_EAUniqueTicketID	The unique ID of the alert in Enterprise Alert®.
TP_EATicketID	The internal ID of the alert used for matching user response messages.
TP_ExternalTicketID	The external 3rd party alert ID, specified by the event provider on raising the initial event
TP_EATicketStatusOld	The old status of the alert
TP_EATicketStatus	The new status of the alert
TP_EAResolvedBy	The name of the alert user who has initiated the alert status change. If the status change was not initiated by a user, the value is empty.
TP_EAResolutionText	Description of the alert status change.
EP_<param Name>	Event parameters of the source event

HTTPRequestType	Specifies the type of request. This is always set to *HTTP_EVENT*

Attachments

Each attachment is added as a file parameter to the HTTP POST.

Response

The Web application should respond with the HTTP error code 200 (*HTTP_OK*) if the request could be handled successfully on the server side. In this case, the event message will marked as transmitted in Enterprise Alert®.

To mark the message as failed, set your error message as the HTTP status description and return an HTTP error code greater than 299 in the HTTP Response (e.g. 500 = Internal Server Error).

10.2.2 Handling of alert status changes in Web services received via HTTP SOAP

For handling alert status changes in SOAP format, a Web service must be implemented which precisely matches the Web service description defined in the WSDL document *EAEventHandler.wsdl*. This document can be found in the folder *C:|Program Files| EnterpriseAlert |Web Connector|WSDL* by default. The WSDL document can be used to generate a skeleton for implementing the Web service in the programming language you want.

The Web service must implement the one and only the method *HandleEAEvent()*.

The following paragraphs provide a detailed description of the Web service method *HandleEAEvent()* and its parameters.

Signature

```
EAResponseType HandleEAEvent (EAEventType eaEvent)
```

Parameter EAEventType

The parameter *EAEventType* of method *HandleEAEvent()* is a complex data type. It consists of the following properties.

Property	Description
ID (int)	ID of the alert event in Enterprise Alert®
Name (string)	Name of the alert event. The value is always *OnTicketStatus*.
Type (string)	Type of the alert event. The value is always *TicketProcessing*.
Timestamp (DateTime)	Timestamp of the event in format *YYYY-MM-DD hh:mm:ss*
EventParameters	An array of the source event parameters as complex data type *ParameterType*

TicketParameters	An array of the alert parameters as complex data type *ParameterType. These parameters can be:* EAUniqueTicketID EATicketID ExternalTicketID EATicketStatusOld EATicketStatus EAResolvedBy EAResolutionText
Attachments	An array of the complex data type *Attachment*

Complex data type ParameterType

ParameterType consists of the following properties.

Property	Description
Name (string)	The name of the event parameter
Value (string)	The value of the event parameter

Complex data type AttachmentType

AttachmentType consists of the following properties.

Property	Description
ID (int)	Unique ID of the attachment.
ContentType (string)	The mime content type of the attachment like text/plain or image/jpg
Encoding (enum)	Encoding of the attachment. Either *noencoding* for text attachments or *base64* for binary attachments
Location (string)	The file name of the attachment
Value (string)	The content of the attachment encoded in the specified encoding

Return value EAResponseType

The return value ***EAEventType*** of method ***HandleEAEvent()*** is a complex data type. It consists of the following properties.

Property	Description
ID (int)	ID of the event to which the response is assigned
Timestamp (DateTime)	Timestamp of the response
ErrorMsg (string)	An error message describing the result

Result (int)	An error code. If the value is zero the event message will be marked as transmitted in Enterprise Alert®, otherwise it will be marked as failed.

11 SAMPLE IMPLEMENTATIONS

Derdack provides a sample Web service and a sample Web application that can be used as a starting point to implement a service that is able to handle alert status change events in Enterprise Alert®. The samples are created with Microsoft Visual Studio 2008 and are written in C#. The implementation of the Web application and the Web service writes the event parameters to a log file.

The sample project can be found in the ***Samples*** folder under the Enterprise Alert® installation folder (by default ***C:|Program Files| EnterpriseAlert |Samples|HTTP SOAP API|***). The project ***EAEventHandler*** contains the Web service in the file ***EAEventHandler.asmx (.cs)*** and the Web application for handling simple post requests in the file ***EAEventHandler.aspx(.cs)***. To activate the samples, you need to copy the project folders to the IIS server root (by default ***C:|Inetpub|wwwroot|)*** and activate the services in the IIS configuration.

The following paragraphs provide basic steps describing how to implement event handlers.

11.1 Developing an event handler Web service for alert status change events

The starting point to develop a Web service for handling alert status change events in Enterprise Alert® is the ***EAEventHandler.wsdl*** document. The document can be found in the ***HTTP/SOAP API*** folder under the ***Samples*** folder (by default ***C:|Program Files| EnterpriseAlert |Samples|HTTP SOAP API|***).

A WSDL (Web service Description Language) document written in XML specifies the methods, data types and protocols of a Web service.

You should be able to find tools for generating server skeletons, client proxies and the complex data types from the WSDL description for each Web service programming language. The server skeletons are classes with declarations of the Web service methods. After the implementation of these methods, the Web service is complete and can be deployed in a Web server. To write a Web service client, the client proxies can be used to interact with the Web service.

The following steps describe how to generate Server Skeletons for ***EAEventHandler*** and how to implement the Web service with Visual Studio 2008.

11.1.1 Step 1: Generating server skeletons with .NET

To generate server skeletons with .NET, you can use the tool wsdl.exe with the option /server. The complete command is:

> wsdl.exe /server /namespace:EAEventHandler EAEventHandler wsdl

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

The C# source code file *EAEventHandlerService.cs* will be generated. This file contains the abstract class *EAEventHandlerService* with the method *HandleEAEvent()*, which has to be implemented by a deriving Web service implementation class.

11.1.2 Step 2: Implementing the Web service in C#

To implement the Web service method, open Visual Studio 2008 and select *New Project*. Select *ASP.NET Web Service Application* in the group *C# Projects*. Enter *EAEventHandler* as project name. After clicking OK, the wizard creates a project.

Right click on the *EAEventHandler* project in the Solution Explorer and select *Add Existing Item*. Next, select the *EAEventHandler.cs* file generated in Step 1 and click OK. The server skeletons and data types are now part of this project.

To implement the abstract class *EAEventHandler* you can use the generated Web service class Service1 in the file Service1.asmx. First, rename the file to *EAEventHandler.asmx*. To do so, right click on the file *Service1.asmx* in the Solution Explorer and select *Rename*. Next, change the class name to *EAEventHandlerServiceImpl*. Switch to the class view, expand the namespace *EAEventHandler*, click right on the class *Service1* and select *Properties*. In the properties window, change the name from *Service1* to *EAEventHandlerServiceImpl*.

To open the code view of the class, double click on *EAEventHandlerServiceImpl* in the class view. Next, import the namespace of the abstract class *EAEventHandlerService* and the complex data types. Add the following *using* statement at the bottom of the existing *using* statements:

```
using EAEventHandler;
```

The derived class inherits from *System.Web.Services.WebService*. To implement the abstract class *EAEventHandlerService*, change the base class to the following:

```
public class EAEventHandlerServiceImpl : EAEventHandlerService
```

Lastly, you will need to override and implement the abstract methods of *EAEventHandlerService* in the derived class.

In C# you have to add the WebMethod attribute to the method HandleEAEvent() in the class EAEventHandlerImpl. The attribute that describes the namespace of the Web service ("HTTP://derdack.com/webservices/EAEventHandler") is also necessary and must be added to the class declaration as well.

11.2 Sample Web service implementation for handling alert status changes

The complete source code of an *EAEventHandlerServiceImpl* sample implementation with class and method attributes and correct return values is displayed below:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;
```

```
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Xml.Serialization;
using EAEventHandler;

namespace EAEventHandler
{
    /// <summary>
    /// Summary description for EAEventHandler
    /// </summary>
    [WebServiceAttribute(Namespace =
            "HTTP://derdack.com/webservices/EAEventHandler")]
    [WebServiceBindingAttribute(
            Name = "EAEventHandlerServiceSOAP",
            Namespace = "HTTP://derdack.com/webservices/EAEventHandler")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    public class EAEventHandlerServiceImpl : EAEventHandlerService
    {

        [WebMethodAttribute()]
        [SOAPDocumentMethodAttribute(
            "HTTP://derdack.com/webservices/EAEventHandler/HandleEAEvent",
            RequestNamespace =
                  "HTTP://derdack.com/webservices/EAEventHandler",
            ResponseNamespace =
                  "HTTP://derdack.com/webservices/EAEventHandler",
            Use = SOAPBindingUse.Literal,
            ParameterStyle = SOAPParameterStyle.Wrapped)]
        [return: XmlElementAttribute("EAResponse",
            Namespace = "HTTP://derdack.org/EATypes.xsd")]
        public override EAResponseType HandleEAEvent(EAEventType EAEvent)
        {
            // TODO handle the event here
            //


            // return the result
            //
            return new EAResponseType()
            {
                ErrorMsg = "",
                ID = EAEvent.ID,
                Result = 0,
                Timestamp = DateTime.Now
            };
        }
    }
}
```

11.2.1 Attachments

The *EAEventType* parameter contains the attachments as an array of *AttachmentType* objects in the property *Attachments*. Each *AttachmentType* object encapsulates an attachment in the four properties *ID*, *Location*, *Encoding* and *Value*. *Value* contains the content of the attachment, *Location* contains the file name and *Encoding* the transfer encoding of the *Value*. Possible values are base64 for binary attachments and noencoding for all others.

The following extract of the web method *HandleEAEvent()* saves all attachments in *EAEventType* as files.

```
foreach (AttachmentType attachment in eaEvent.Attachments)
{
	// write attachments to files
	if (attachment.Encoding == AttEncodingType.base64)
	{
		byte[] byteContent = Convert.FromBase64String(attachment.Value);
		File.WriteAllBytes("C:\\" + attachment.Location, AttContent);
	}
	else
	{
		string strContent = attachment.Value;
		File.WriteAllText("C:\\" + attachment.Location, strContent);
	}
}
```

11.3 Sample Web application for handling alert status changes via HTTP POST request

Open Visual Studio 2008 and select *New Project*. The wizard opens. Select A*SP.NET Web Application* from the group *C# Projects*. Enter *EAEventHandlerWebApp* as a project name. After selecting OK, the wizard will generate the project.

Firstly, you need to rename the file *WebForm1.aspx* as *EAEventHandlerForm.aspx*. To do so, right click on the file *WebForm1.aspx* in the Solution Explorer and select *Rename*. Now rename the class *WebForm1* as *EAEventHandlerForm*. Switch to the class view, expand the namespace *EAEventHandlerWebApp*, right click the class *WebForm1* and select *Properties*. In the properties window, change the name from *WebForm1* to *EAEventHandlerForm*.

The method *Page_Load* will be called if the Web client of Enterprise Alert® requests the *EAEventHandlerForm*. Here you have to implement the handler functionality. The event parameters are stored in the member variable *Request*.

11.3.1 Page_Load method

```
protected void Page_Load(object sender, EventArgs e)
{
	foreach (string strParam in Request.Params.AllKeys)
	{
	string strValue = Request.Params[strParam];

		// TODO process the event parameters
		//

		if (strParam == "EventID")
		{
			// …
		}
	}

	// save the attachments
	//
	HTTPFileCollection Attachments = Request.Files;
```

```
        if ((Attachments != null) && (Attachments.Count > 0))
        {
            foreach (string strHTTPFileKey in Attachments.Keys)
            {
                HTTPPostedFile HTTPFile = Attachments[strHTTPFileKey];
                if (HTTPFile.ContentLength == 0)
                    continue;

                string strFileName = HTTPFile.FileName;

                byte[] fileContent = new byte[HTTPFile.ContentLength];
                HTTPFile.InputStream.Read(fileContent, 0,
                    fileContent.Length);

                File.WriteAllBytes("C:\\" + strFileName, fileContent);
            }
        }

        // Response.StatusCode = 500; // on error

        Response.Write("OK");
}
```

12 SOAP CONNECTOR SDK INTRODUCTION

In addition to the HTTP/SOAP API, which provides a standard interface for 3rd party event providers defined by Derdack, Enterprise Alert® comes with a fully flexible template based application interface called the SOAP Connector SDK.

To apply this interface for the request/response format of individual 3rd party event providers, templates in the form of an XSL stylesheet or .NET assemblies can be installed. These templates manage the conversion of HTTP GET, HTTP POST or SOAP messages to the internal XML message format of Enterprise Alert® and vice versa.

13 ARCHITECTURE

The SOAP Connector SDK is split into a Web server and Web client.

The Web server is installed in the local IIS and the main page can be requested through the URL:

http://localhost/EAWebService/EventConnectorServer.ashx

For each individual 3rd party event provider, a *handler* template needs to be installed for handling according HTTP requests and for forwarding them as event messages to the messaging service of Enterprise Alert®.

A 3rd party event provider can address the server template by adding the template name as a parameter to the URL:

http://localhost/EAWebService/EventConnectorServer.ashx?Handler=[template name]

If the alert status for an alert previously triggered by an event provider changes, the Web client of the SOAP Connector SDK will call the according template to forward the status update to the event provider as a HTTP request.

14 ABOUT TEMPLATE PACKS

14.1 Structure of a template pack

A template pack is a zip archive containing all the files needed for installing and running the pack in the SOAP Connector SDK.

The content of the zip file must have the following structure:

- The folder WebHandler contains one or more server template files required for handling, translating and forwarding the HTTP requests of 3rd-party event providers to the messaging service of Enterprise Alert®.

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

This can take the form of .NET assemblies, in which case one of the assemblies must contain a .NET class with a main entry point. Otherwise, an XSL stylesheet can be used.

- The folder WebRequester contains client template files required for handling alert status changes in Enterprise Alert® and for forwarding these status updates to the registered event provider URL. This can take the form of .NET assemblies, in which case one of the assemblies must contain a .NET class with a main entry point. Otherwise, an XSL stylesheet can be used.
- manifest.xml stores installation information for the template.

14.2 The Manifest

The manifest specifies properties for installing the template pack in the SOAP Connector SDK. It needs to be stored as manifest.xml in the root of the template pack zip archive. The manifest must have the following format:

```
<?xml version="1.0"?>
<TemplatePack
      xmlns:xsi="HTTP://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="HTTP://www.w3.org/2001/XMLSchema">
  <Name />
  <Description />
  <WebHandler>
      <Name />
  </WebHandler>
  <WebRequester>
      <Name />
      <TargetURI />
      <RequestTimeout />
      <ResendAttempts />
  </WebRequester>
</TemplatePack>
```

14.2.1 Root element TemplatePack

Element *TemplatePack* contains the following child elements:

- *Name*: The name of the template. Event providers have to specify this name in the handler parameter of the request URL on sending requests to the template.
- *Description*: A description of the template.
- *WebHandler*: Parent element for defining properties of the request handler part of the template.
- *WebRequester*: Parent element for defining properties of the requester part of the template.

14.2.2 Element WebHandler

Element *WebHandler* contains the following child elements:

- *Name*: The name of the handler for the web handler template, which can be either the name of the .NET class containing the main entry point or the name of an XSL stylesheet file. If the handler is a .NET class, then the full name of the class including the namespace must be specified e.g. *namespace.classname*. The class must exist in one of the .NET assemblies in the ***WebHandler*** folder of the template pack. If the handler is an XSL stylesheet, then the stylesheet file must exist in the root of the WebHandler folder.

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

14.2.3 Element WebRequester

Element *WebRequester* contains the following child elements:

- *Name*: The name of the handler for the web requester template, which can be either the name of the .NET class containing the main entry point or the name of an XSL stylesheet file. If the handler is a .NET class, then the full name of the class including the namespace must be specified e.g. *namespace.classname*. The class must exist in one of the .NET assemblies in the WebRequester folder of the template pack. If the handler is an XSL stylesheet, then the stylesheet file must exist in the root of the WebRequester folder.
- *TargetURI*: The URI of the 3rd party event provider which is able to handle the request of this template if a alert status update will be forwarded to the event provider.
- *RequestTimeout*: Maximum time to wait for the server to respond to a request. If this time is exceeded, a resend will be performed or the event status will be set to failed in Enterprise Alert®.
- *ResendAttempts*: Number of resend attempts if an error occurs on sending a request to the 3rd-party event provider.

14.2.4 Sample template pack

In the following sample we create a template pack for a 3rd party event provider which raises its events via HTTP GET requests. The template pack should be named ***HTTP GET***. Because of this name, the event provider has to use the URL http://localhost/EAWebService/EventConnectorServer.ashx?Handler=HTTP GET for sending requests to this template.

The main entry point of the server template is the .NET class ***HTTPGetHandler*** in the namespace ***MyEventProvider***. This class is stored in the .NET assembly ***HTTPGetHandler.dll***.

The event provider is able to handle HTTP POST requests for alert status changes in Enterprise Alert®. The response address used is http://localhost/EAEventHandler/EAEventHandler.aspx. The client template is implemented as a XSL stylesheet named HTTPPostSender.xsl.

We can create the following manifest from the above information:

```
<?xml version="1.0"?>
<TemplatePack
      xmlns:xsi="HTTP://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="HTTP://www.w3.org/2001/XMLSchema">
  <Name>HTTP GET</Name>
  <Description>Event provider template for HTTP GET method</Description>
  <WebHandler>
    <Name>MyEventProvider.HTTPGetHandler</Name>
  </WebHandler>
  <WebRequester>
    <Name>HTTPPostSender.xsl</Name>
      <TargetURI>
            http://localhost/EAEventHandler/EAEventHandler.aspx
      </TargetURI>
    <RequestTimeout>42</RequestTimeout>
    <ResendAttempts>2</ResendAttempts>
  </WebRequester>
</TemplatePack>
```


PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

In the last step we have to create the resulting *HTTPGetTemplate.zip* archive with the following structure:

- \ WebHandler \ HTTPGetHandler.dll
- \ WebRequester \ HTTPPostSender.xsl
- \ manifest.xml

Now we can install the template pack in Enterprise Alert®.

14.3 Template packs in the web portal of Enterprise Alert®

14.3.1 Installation of template packs

To install a template pack, open the Web portal of Enterprise Alert® and log on as an administrator.

Navigate to the Event Filter and click *Setup Source from Template*. In the opened dialog select the link *Upload zip file* and choose the template pack you want to install. If the upload was successful, select the link *Install*.

During installation the template pack will be verified. If the template pack is invalid, an error will be displayed and the installation aborted.

If the installation was successfully, the template will be displayed as a new event source in the left hand event source tree of the Event Filter. You can then add a filter to this event source to create an alert or to send messages.

14.3.2 Reloading templates

The server template files inside the *WebHandler* folder of the template pack zip archive will be copied to the folder *C:|inetpub|wwwroot|EAWebService|WebHandler|[template name] by default during installation*. Similarly, the client template files inside the WebRequester folder will be copied to the folder *C:|Program Files|EnterpriseAlert |WebConnector|WebRequester|[template name]*.

If you wish to modify the templates and test the changes immediately, you can simply replace the existing files with your modified ones at run time. To reload the modified files on the server, first open the Web portal of Enterprise Alert® and log on as an administrator. Then navigate to the Event Filter and right click on the corresponding event source in the left hand event source tree. In the context menu select *Reload Template and Filters*.

The template will be then reloaded on the server. If the reload is unsuccessful, an error message with detailed information will be displayed.

14.3.3 Downloading existing template packs

For downloading the existing template pack of an event source, first open the Web portal of Enterprise Alert® and log on as an administrator. Navigate to the Event Filter and right click on the corresponding event source in the left hand tree. In the context menu select *Download Template*. Select a folder where you would like to store the template pack and select *Save*.

The template pack will be downloaded to the specified location. You can then modify and re-install the template pack as required.

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

If you wish to download a blank template pack as a starting point for developing your own template, select *Download Blank Template Pack* in the Event Filter.

15 CONNECTOR TEMPLATE DEVELOPMENT

15.1 XSL stylesheet or .NET class?

The advantage of XSL stylesheet files is that you can simply modify a template and reload it without compiling. The disadvantage is the limited functionality. If you want to simply translate XML text into another format and you are familiar with the internal XML format used by Enterprise Alert®, then you can create XSL stylesheet files as templates. If you are not familiar with the XML format or you need more complex functionality e.g. database access or complex arithmetic, you should create your template as a .NET class as part of a .NET assembly.

15.2 Template requirements

Regardless of whether the template comes in the form of .NET class or an XSL stylesheet file, the functionality provided is the same.

Firstly, the server templates have to translate HTTP requests from 3rd party event providers into messages or function calls which can be handled by the SOAP Connector SDK, and secondly, the results need to be translated into corresponding HTTP responses. Furthermore, the templates have to provide meta-information about the possible events and their parameters. This meta-information will be requested by the SOAP Connector SDK upon installation of a template and is then stored for use in the Event Filter of Enterprise Alert®.

The web requester template translates messages or function calls concerning alert status changes in Enterprise Alert® to HTTP requests interpretable by the 3rd party event provider as well as HTTP responses to corresponding status messages or function calls.

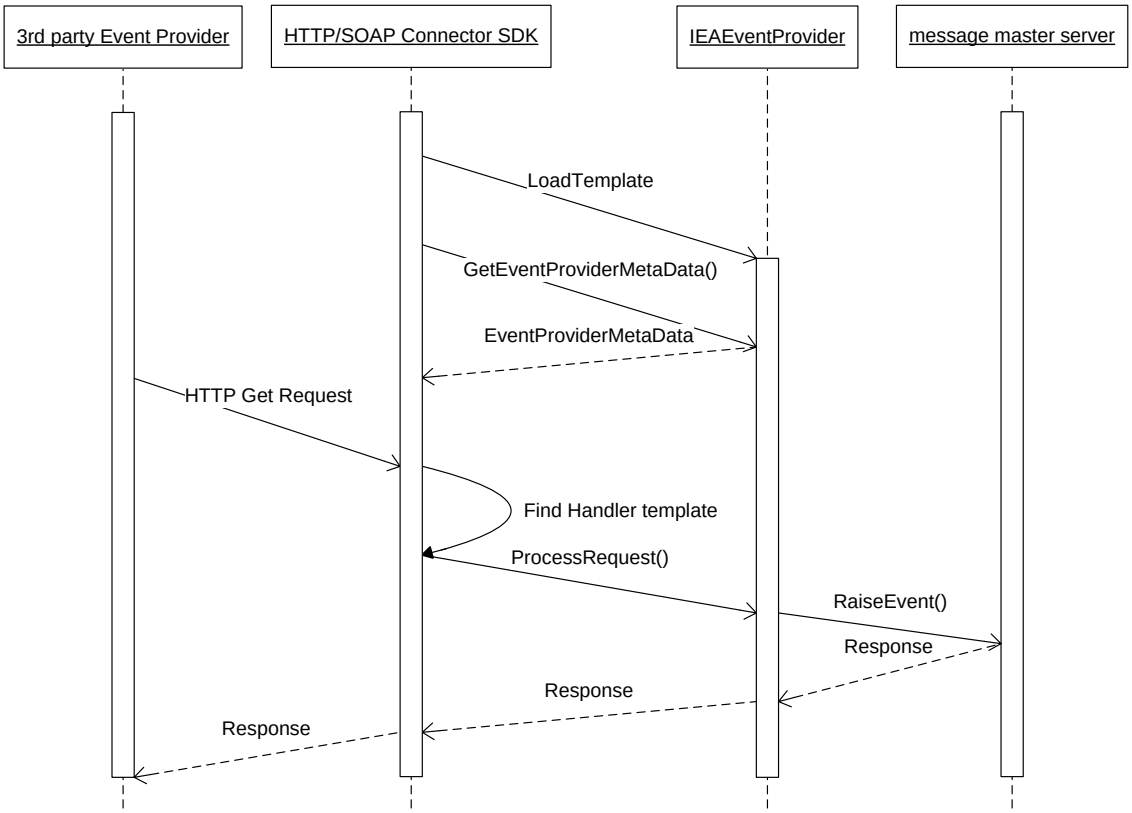
15.3 .NET classes as templates

15.3.1 Server templates need to implement IEAEventProvider

A server template written as a .NET class needs to implement the interface *IEAEventProvider*. In order to provide meta-data about event parameters, the template needs to implement the method *GetEventProviderMetaData(). GetEventProviderMetaData()* will be called by the SOAP Connector SDK after loading the template class. The return value describes the event parameters that will be set when the event is raised. If *GetEventProviderMetaData()* is called successfully when the template is loaded, you will be able to see the according event provider entry in the Event Filter of Enterprise Alert®. You will then be able to create event filters for the event provider. The event parameters as returned by *GetEventProviderMetaData()* can be used as filter criteria or for composing the alert text.

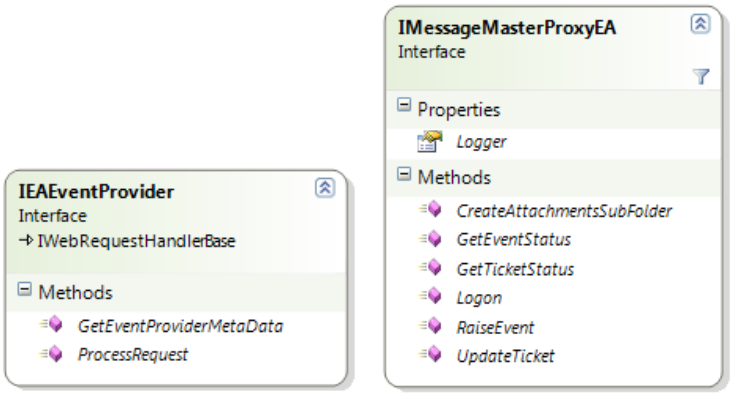
The second method of the interface is *ProcessRequest()*. This must be implemented to handle the HTTP requests from the 3rd party event provider. The following sequence diagram illustrates the workflow in the SOAP Connector SDK when loading a server template and when requests from event providers are forwarded to it:

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585



Loading server template and calling handler method on HTTP request

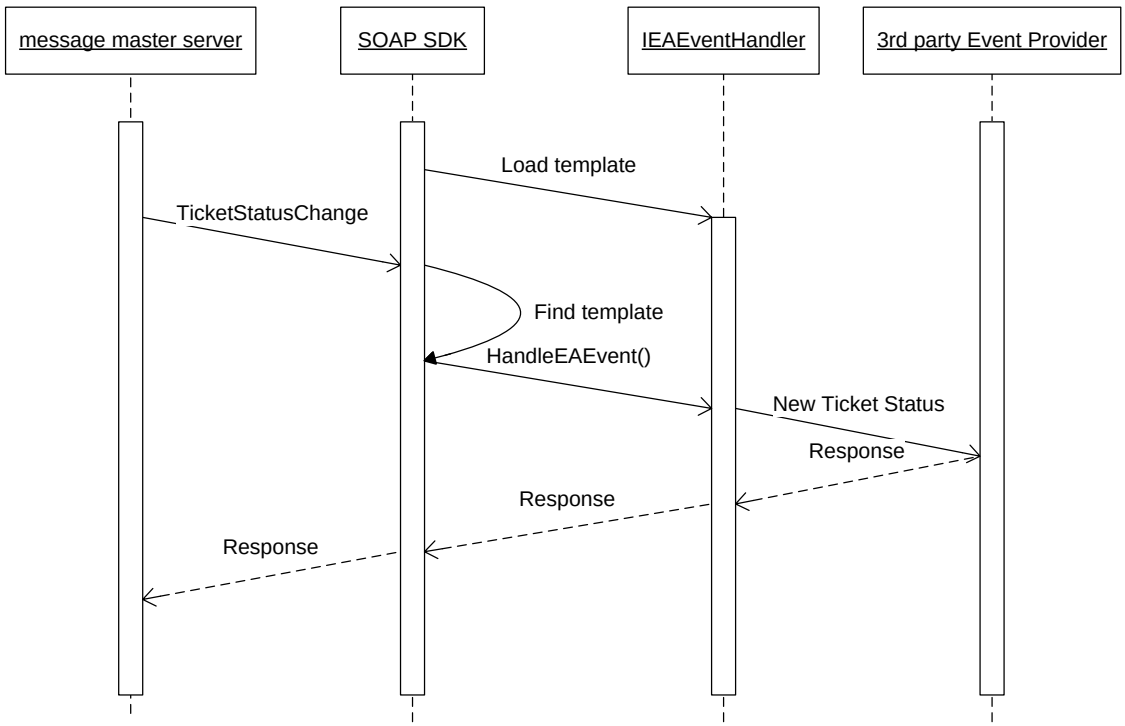
An instance of ***IMessageMasterProxyEA*** is provided as a parameter to the ***ProcessRequest()*** method. This object provides functionality for doing actions in Enterprise Alert® e.g. the raising of events. The second parameter is the current ***HTTPContext*** object which contains the HTTP request object that can be used for authenticating and processing the request.



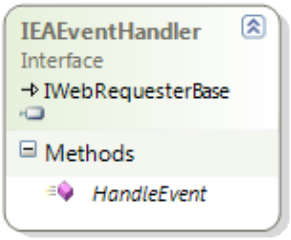
IEAEventProvider and IMessageMasterProxyEA

15.3.2 Client templates need to implement IEAEventHandler

Client templates that forward alert status changes to external event providers need to implement the interface *IEAEventHandler*. The single method available in the interface namely ***HandleEvent(),*** will be called by the SOAP Connector SDK if an alert triggered by an event raised by the according event provider changes its status. The following sequence diagram illustrates the workflow in the SOAP Connector SDK when loading a client template and forwarding alert status changes from Enterprise Alert® to the template.



HandleEvent() has one parameter of type *EAEvent*. This object contains information about the old and new alert statuses as well as the alert ID. This information is useful to the event provider and should be forwarded by the template in the HTTP request to the event provider.



IEAEventHandler

15.4 XSL stylesheet files as templates

This chapter describes how XSL stylesheet files can be used as templates in the SOAP Connector SDK. In addition, the XML format of Enterprise Alert® event messages and status messages will be explained.

The snippets in the following sections are based on the XSL stylesheet files of the *HTTP POST* sample template inside the blank template pack. For the complete XSL stylesheet files, you can download the blank template pack from the Enterprise Alert® Web portal and extract *HttpPostTemplate.zip* from the downloaded zip archive.

15.4.1 Providing event parameter meta data for the XSL server template

As the first part of the XSL server template, possible event parameters need to first be populated in the SOAP Connector SDK.

To do this the XSL template must contain the following XML fragment:

```
<xsl:template match="mm_eventprovider_meta">
    <EAEventProviderMetaData>
      <Parameters>
        <Parameter>
          <Name>Param1</Name>
          <DisplayName>Parameter 1</DisplayName>
          <Description>This is Parameter 1</Description>
        </Parameter>
        <Parameter>
          <Name>Param2</Name>
          <DisplayName>Parameter 2</DisplayName>
          <Description>This is Parameter 2</Description>
        </Parameter>
      </Parameters>
    </EAEventProviderMetaData>
  </xsl:template>
```

You need to specify all parameters as *param* elements in your template. These parameters can then be forwarded when raising an event in Enterprise Alert®.

15.4.2 XSL server template for events raised by HTTP requests

A message for raising an event in Enterprise Alert® has the following XML structure:

```
<mm_message>
      <mm_header type="event" subtype="inbound"/>
      <mm_event name="NewConnectorEvent" type="ConnectorEvent">
          <param name="Param1">
            <value/>
          </param>
          <param name="Param2">
            <value/>
          </param>

              ...
      </mm_event>
 </mm_message>
```

Received HTTP POST or GET requests will be passed to the XSL transform in the following form. The *param* elements in the XML are taken from the parameters of the HTTP request:

```
<mm_webRequest>
      <param name="Param1">
        <value/>
      </param>
      <param name="Param2">
        <value/>
      </param>

              ...
</mm_webRequest>
```

HTTP requests in XML format like SOAP will be passed to the XSL translator as received without modification. In this case the XSL stylesheet file needs to directly transform the XML into the event message format of Enterprise Alert®.

To raise an event in Enterprise Alert® from the received Web request, we simply transform the input XML to the XML of an event message using the following XSL template:

```
<xsl:template match="mm_webRequest">
    <mm_message>
      <mm_header type="event" subtype="inbound" />
      <mm_event name="NewConnectorEvent" type="ConnectorEvent">
         <param name="Param1">
            <value>
               <xsl:value-of select="param[@name='Param1']/value"/>
            </value>
         </param>
         <param name="Param2">
            <value>
              <xsl:value-of select="param[@name='Param2']/value"/>
```


PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

```
            </value>
          </param>
        </mm_event>
      </mm_message>
</xsl:template>
```

Enterprise Alert® will immediately respond with the following status message.

```
<mm_message>
      <mm_header index="123456" type="status" subtype="inbound" .../>
      <mm_status status="transmitted" resultcode="0">
            <description>OK</description>
      </mm_status>
 </mm_message>
```

The status message needs to then be transformed to a Web response, also in XML format, for sending back to the HTTP client.

```
<xsl:template match="mm_message">
      <mm_webResponse contentType="text/xml">
          <RaiseEventResult>
            <xsl:choose>
              <xsl:when test="mm_status/@status='transmitted'">
                <EventID>
                  <xsl:value-of select="mm_header/@index"/>
                </EventID>
              </xsl:when>
              <xsl:otherwise>
                <Status>
                  <xsl:value-of select="mm_status/@status"/>
                </Status>
                <StatusCode>
                  <xsl:value-of select="mm_status/@resultcode"/>
                </StatusCode>
                <Description>
                  <xsl:value-of select="mm_status/description"/>
                </Description>
              </xsl:otherwise>
            </xsl:choose>
          </RaiseEventResult>
        </mm_webResponse>
</xsl:template>
```

The client then receives a response in the following format:

```
<RaiseEventResult>
      <EventID>123456</EventID>
</RaiseEventResult>
```

If Enterprise Alert® responds with a failed status, the HTTP client will receive a response in the following format:

```
<RaiseEventResult>
```


PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

```
      <Status>
            failed
      </Status>
      <StatusCode>
            -1
      </StatusCode>
      <Description>
            Destination not supported
      </Description>
</RaiseEventResult>
```

The complete XSL stylesheet file needs to be installed as a server template in the SOAP Connector SDK.

15.4.3 XSL client template for forwarding alert status changes as HTTP requests

Enterprise Alert® sends alert status updates as event messages with type *TicketProcessing* and name *OnTicketStatus*. The message contains the new and old statuses of the alert as well as the alert ID and external ID.

```
<mm_message>
      <mm_header type="event" subtype="inbound" index="123456" />
      <mm_event name="OnTicketStatus" type="TicketProcessing">
          <param name="ID">
            <value/>
          </param>
          <param name="InternalID">
            <value/>
          </param>
          <param name="ExternalID">
            <value/>
          </param>
          <param name="OldStatus">
            <value/>
          </param>
          <param name="NewStatus">
            <value/>
          </param>
          <param name="MemberName">
            <value/>
          </param>
          <param name="Description">
            <value/>
          </param>
      </mm_event>
 </mm_message>
```

For instance the alert event message needs to be transformed to a HTTP POST Web request before sending it to the event provider. The following XSL template can be used to create the request:

```
<xsl:template match="mm_message">
    <xsl:if test="mm_header[@type='event'] and
                  mm_event[@name='OnTicketStatus']">
      <mm_webRequest
```

```
        xslResultFormat="Params"
        method="POST"
        contentType="multipart/form-data"
        attachmentFormat="FileUpload"
        encodeBinaryAttachments="Base64"
      >
        <param type="text" name="EventID">
          <value>
            <xsl:value-of select="mm_header/@index"/>
          </value>
        </param>
        <param type="text" name="EventName">
          <value>
            <xsl:value-of select="mm_event/@name"/>
          </value>
        </param>
        <param type="text" name="EventType">
          <value>
            <xsl:value-of select="mm_event/@type"/>
          </value>
        </param>
        <xsl:for-each select="mm_event/param">
          <param type="text">
            <xsl:attribute name="name">
                  EventParam<xsl:value-of select="@name"/>
            </xsl:attribute>
            <value>
              <xsl:value-of select="value"/>
            </value>
          </param>
        </xsl:for-each>
        <xsl:for-each select="mm_meta/item">
          <param type="text">
            <xsl:attribute name="name">
                  TicketParam<xsl:value-of select="@name"/>
            </xsl:attribute>
            <value>
              <xsl:value-of select="value"/>
            </value>
          </param>
        </xsl:for-each>
      </mm_webRequest>
    </xsl:if>
  </xsl:template>
```

The resulting HTTP request can be customized through the attributes of the root element *mm_webRequest*. The following attributes and values are possible:

- *method*
  Specifies the HTTP request method. Possible values are:

Value	Description
POST	The Web request will be send as an HTTP POST request.

GET	The Web request will be send as an HTTP GET request.

- *xslResultFormat*
  Specifies the format of the content after XSL transformation. Possible values are:

Value	Description
Params	The result contains a set of ***param*** elements whose values should be sent as HTTP request parameters.
XML	The result is in XML and will be sent in the HTTP request body as is. The method must be set to POST.
Soap	The result is SOAP and will be sent in the HTTP request body as is. The method must be set to POST.
PlainText	The result is plain text. If method is set to POST, the content will be sent in the HTTP request body. If the method is set to GET, the content will be appended to the request URI.

- *contentType*
  Specifies the content type of the HTTP request. Possible values are:

Value	Description
application/x-www-form-urlencoded	The request parameters will be added as URL encoded key-value pairs. This value can be used in HTTP GET and HTTP POST requests.
multipart/form-data	The request parameters will be added to the POST content in multipart mime format. For each parameter a MIME part will be added. Attachments will be added as file parameters. This value can be used in HTTP POST requests only.
multipart/mixed or multipart/related	The resulting content of the XSL transformation will be added as the first mime part. Attachments will be added as additional multipart MIME to the second MIME part. The values can be used in HTTP POST requests only.
text/plain or text/xml	The resulting content of the XSL transformation will be added as content to the HTTP POST request if the method has been set to POST. For GET requests, the XSL result will be appended to the request URI.

- *attachmentFormat*
  Specifies how the attachments will be added to the HTTP POST request. Possible values are:

Value	Description
None	Attachments will not be added to the HTTP request.

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

FileUpload	Attachments will be added as file upload parameters to the POST request.
MultipartMime	Attachments will be added as multipart mime to the second mime part of the POST content.
Dime	Attachments will be added in Microsoft DIME format. This can be used Microsoft Web services are called.

- encodeBinaryAttachments:
  Specifies how binary attachments should be encoded. Possible values are:

Value	Description
None	Binary attachments will not be encoded.
Base64	Binary attachments will be encoded as base 64.

- *addStartParam*
  Specifies whether a start parameter should be added to attachments in multipart MIME format. Possible values are:

Value	Description
true	Start parameter will be added.
false	Start parameter will not be added.

- *httpUserName* and *httpPassword*
  If these values are specified, the request will be sent using HTTP basic authentication.

- *soapAction*
  The specified value is set in the HTTP POST header parameter ***SOAPAction*** if the XSL transformation result is SOAP.

The sample XSL transformation creates an HTTP POST request in multipart/form-data format, where the attachments are added as file upload parameters.

After submitting the POST request to the event provider, the provider responds with OK provided the request could be processed successfully, otherwise an error message is returned. To transform the HTTP response of the provider to the format of a Enterprise Alert® status message, we use the following XSL template:

```
<xsl:template match="mm_webResponse">
    <mm_message>
      <mm_header type="status" subtype="inbound">
      </mm_header>
      <mm_status>
        <xsl:choose>
          <xsl:when test="contains(string(.),'OK')">
            <xsl:attribute name="status">transmitted</xsl:attribute>
```


PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

```
            <xsl:attribute name="resultcode">0</xsl:attribute>
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name="status">failed</xsl:attribute>
            <xsl:attribute name="resultcode">-1</xsl:attribute>
            <description>
              <xsl:value-of select="."/>
            </description>
          </xsl:otherwise>
        </xsl:choose>
      </mm_status>
    </mm_message>
  </xsl:template>
```

The event's status changes to *transmitted* if the event provider responded with OK, otherwise the status is set to failed.

15.5 Implement an event provider template with handlers as .NET classes

The following chapters describe how to implement an event provider template that is able to handle requests from 3rd party event providers via HTTP GET. Alert status changes can be forwarded by the template to the providers via HTTP GET as well. For implementing the handlers, .NET classes should be used.

15.5.1 Download blank template pack

As a starting point for developing your own template pack, you can download a blank template pack from the Enterprise Alert® Web portal. To do so, open the Web portal and log on as an administrator. Navigate to the Event Filter and select *Download Blank Template Pack*.

After extracting the downloaded zip file, you will find three complete working template packs and the source code for a Visual Studio 2008 solution:

- *HTTPSOAPTemplate.zip*
  This template pack contains a .NET assembly used to forward special Web service calls from a 3rd party event provider as event messages to Enterprise Alert®. When alert statuses change in Enterprise Alert, the client templates forwards these change events as SOAP requests to the Web service of the 3rd party event provider.

- *HTTPGetTemplate.zip*
  This template pack contains a .NET assembly used to forward simple Web calls from a 3rd party event provider via HTTP GET as event messages to Enterprise Alert®. When alert status change in Enterprise Alert, the client template forwards these change events as HTTP GET requests to the Web application of the 3rd party event provider.

- *HTTPPostTemplate.zip*
  This template pack contains an XSL stylesheet file used to forward simple Web calls from a 3rd party event provider via HTTP POST as event messages to Enterprise Alert®. When alert status changes in Enterprise Alert, the client template forwards these change events as HTTP POST requests to the Web application of the 3rd party event provider.

PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

- *Source.zip*
  This Zip archive contains the source code of the three template packs and a working Visual Studio 2008 solution.

15.5.2 Downloading and opening the project in Visual Studio 2008

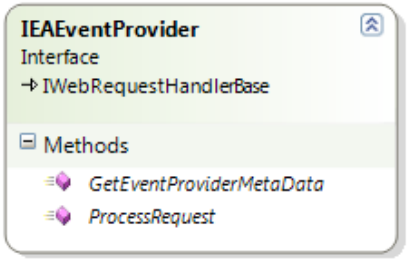
After downloading the blank template pack, extract the downloaded zip file and the *Source.zip* file inside of the blank template pack. Open *EventProviderBlankTemplate.sln* with Visual Studio 2008.

You will find two projects in the solution. The first project *EventProviderBlankTemplateClient* contains three client templates for forwarding alert status changes to the 3rd party Web application. The second project *EventProviderBlankTemplateServer* contains three server templates for forwarding Web requests fr 3rd party event providers as event messages to Enterprise Alert®.

15.5.3 Implementing a server template as a .NET class

Open the file *HTTPGetHandler.cs* from the project *EventProviderBlankTemplateServer* in Visual Studio.

The class *HTTPGetHandler* implements the interface *IEAEventProvider*. This is required for server templates that are loaded by the SOAP Connector SDK of Enterprise Alert®.



Implementation of the interface method GetEventProviderMetaData()

When the sample *HTTPGetHandler* template raises an event it sets two parameters in the event message i.e. *MyParam1* and *MyParam2*. These parameters are returned as meta information for the *HTTPGetHandler* template in the return value of method *GetEventProviderMetaData()*:

```
public EAEventProviderMetaData GetEventProviderMetaData()
{
    return new EAEventProviderMetaData()
    {
        Parameters = new EAEventProviderMetaData.Parameter[]
        {
            new EAEventProviderMetaData.Parameter()
            {
                DisplayName = "My first Parameter",
                Name = "MyParam1",
                Description = "MyParam1 is my first Parameter"
            },
            new EAEventProviderMetaData.Parameter()
            {
                DisplayName = "My second Parameter",
                Name = "MyParam2",
                Description = "MyParam2 is my second Parameter"
```

```
                    }
                }
            };

        }
```

Implementation of the interface method ProcessRequest()

ProcessRequest () is called by the SOAP Connector SDK if a request addressed to the template of the 3rd party event provider is received. This happens when a web client sends a request to the following URI:

HTTP://localhost/EAWebService/EventConnectorServer.ashx?Handler=<template name>

The handler parameter is used by the SOAP Connector SDK to forward the request to the correct template, whereby ***ProcessRequest()*** is called for the corresponding template. In our case, ***ProcessRequest()*** in ***HTTPGetHandler*** will be called.

The implementation of ***ProcessRequest()*** in ***HTTPGetHandler*** extracts all parameters from the URI and performs the action which is specified in the action parameter.

To raise an event via ***HTTPGetHandler***, the following URI must be requested by the 3rd party Web client:
HTTP://localhost/EAWebService/EventConnectorServer.ashx?Handler=HTTP%20Get&action=RaiseEvent&Username=Administrator&Password=&MyParam1=a&Myparam2=b

As a second argument of the method ***ProcessRequest()*** the ***HTTPContext*** will be given to the handler class. The ***HTTPGetHandler*** will read the parameters from the request URL. The request URL can be extracted from the ***HTTPContext.Request*** object: ***HTTPContext.Request.Url***.

The first argument of ***ProcessRequest()*** is an object of type ***IMessageMasterProxyEA***. It is used to perform actions in Enterprise Alert®. It provides methods for raising events, requesting alert statuses and handling messages with attachments.

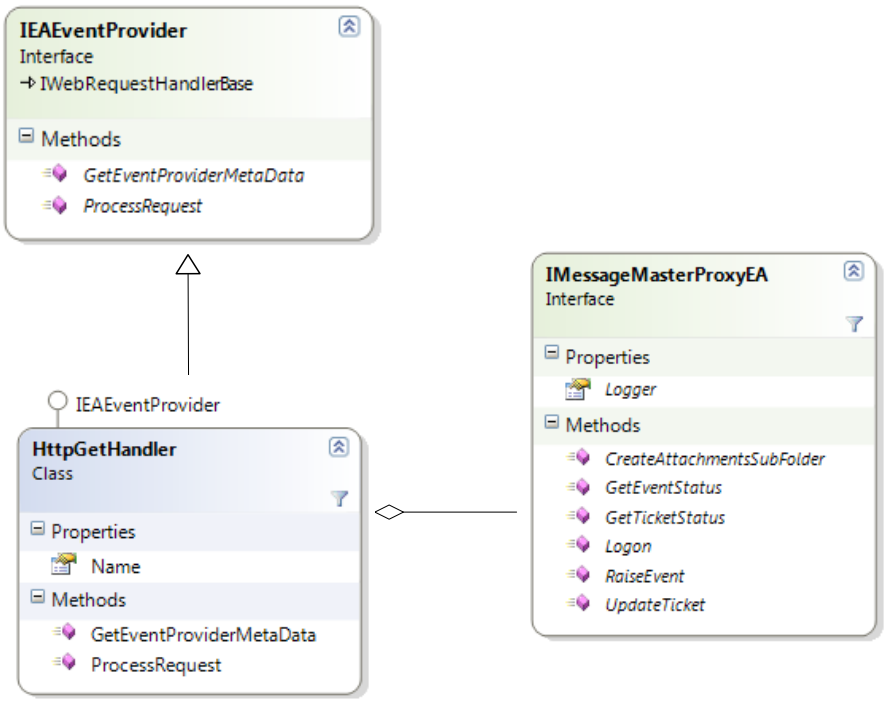
The ***HTTPGetHandler*** uses a proxy the implements ***ImessageMasterProxyEA*** for raising an event:

```
// extract the GET parameters and raise an event in Enterprise Alert
//

string strMyParam1 = colParams["myparam1"];
string strMyParam2 = colParams["myparam2"];

EAEventParameter[] eventParams = new EAEventParameter[]
{
      new EAEventParameter() { Name="myparam1", Value=strMyParam1 },
      new EAEventParameter() { Name="myparam2", Value=strMyParam2 }
};

// log on
mmProxy.Logon(colParams["username"], colParams["password"]);

EAEvent eaEvent = new EAEvent()
{
      EventParameters = eventParams,
      Attachments = colParams["attachments"]
};
```

```
EAEventRaiseResult result = mmProxy.RaiseEvent(this, eaEvent);


// write the result to the HTTP response
//
HTTPContext.Response.Write(result.ToString());
```

Class architecture overview

HTTPGetHandler implements the interface ***IEAEventHandler*** and uses the provided object of type ***IMessageMasterProxy*** in the implementation of ***ProcessRequest()*** to perform actions in Enterprise Alert®.

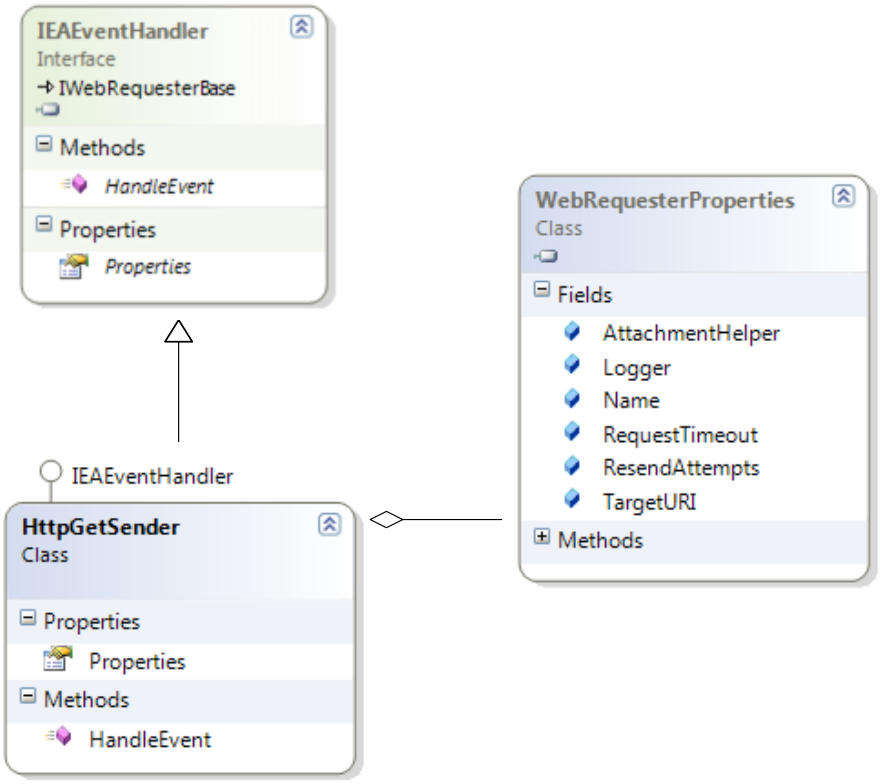


15.5.4 Implementation of a client template as a .NET class

Open the file ***HTTPGetHandler.cs*** from the project ***EventProviderBlankTemplateClient*** in Visual Studio.

The implementation of the interface ***IEAEventHandler*** is required for client templates that are loaded by the SOAP Connector SDK of Enterprise Alert®. Because of this, ***HTTPGetSender*** implements the interface ***IEAEventHandler***.

The interface declares the property named ***Properties*** through which useful settings are provided to the handler class. The only method declared in the interface is ***HandleEvent()***. It must be implemented for handling alert status changes in the ***HTTPGetSender*** class.



The implementation in *HTTPGetSender* is as follows. An *EAEvent* object is given in the parameter of the method *HandleEvent()*. It contains the information about the alert status changes. Because the handler should forward the alert status change as an HTTP GET request, the request parameters must be added to the URI of the external event provider. The URI of the provider is given in the *Properties* member of the *HttpGetSender* object. The event parameters are given in the *EAEvent* object.

The following code extract shows how the Web request will be created and sent to the event provider. Furthermore you can see how the response of the provider is handled and how the *EAEventHandlerResult* is returned to Enterprise Alert®.

```
public EAEventHandlerResult HandleEvent(EAEvent eventToHandle)
{
    try
    {
        // do logging
        Properties.Logger.LogDebug(this, "CreateWebRequest",
                  Properties.Name, Properties.TargetURI);
        Properties.Logger.LogDebug(this, "CreateWebRequest",
                  Properties.Name, eventToHandle.ToString());

        // create the request URI with get parameters
        string strUri = Properties.TargetURI;

        strUri += "?id=" + eventToHandle.ID.ToString();
        strUri += "&name=" + eventToHandle.Name;
        strUri += "&type=" + eventToHandle.Type;
```

```
        foreach (EAEventParameter param in eventToHandle.EventParameters)
        {
            strUri += "&EventParam_" + param.Name + "=" + param.Value;
        }
        foreach (EAEventParameter param in eventToHandle. TicketParameters)
        {
            strUri += "&TicketParam_" + param.Name + "=" + param.Value;
        }


        // create and send request
        WebRequest request = WebRequest.Create(strUri);
        WebResponse webResponse = request.GetResponse();

        // handle the response
        if (webResponse is HttpWebResponse)
        {
            HttpWebResponse httpResponse = (webResponse as HttpWebResponse);
            if (httpResponse.StatusCode != HttpStatusCode.OK)
            {
                return new EAEventHandlerResult()
                {
                    ErrorCode = (int)httpResponse.StatusCode,
                    ErrorText = httpResponse.StatusDescription
                };
            }

            return new EAEventHandlerResult()
            {
                ErrorCode = 0,
                ErrorText = "OK"
            };
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

15.5.5 Creating a template pack

After compiling the .NET assemblies for the server and client template, we need to put them together with the manifest file into a zip package.

First create the folder *MyTemplatePack*. Inside the folder, create the two subfolders *WebRequester* and *WebHandler*. Place your compiled server template assembly in the *WebHandler* folder. Likewise, place your compiled client template assembly in the *WebRequester* folder. In the *MyTemplatePack* root folder, create the file manifext.xml with the following content.

```
<?xml version="1.0"?>
<TemplatePack
      xmlns:xsi="HTTP://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="HTTP://www.w3.org/2001/XMLSchema">
```

```
  <Name>HTTP GET</Name>
  <Description>Event provider template for HTTP GET method</Description>
  <WebHandler>
    <Name>MyEventProvider.HTTPGetHandler</Name>
  </WebHandler>
  <WebRequester>
    <Name>MyEventResponseHandler.HttpGetSender</Name>
    <TargetURI>HTTP://providerhost/EAEventHandler/EAHandler.aspx</TargetURI>
    <RequestTimeout>42</RequestTimeout>
    <ResendAttempts>2</ResendAttempts>
  </WebRequester>
</TemplatePack>
```

Now create a zip archive with the content of the root folder *MyTemplatePack*.

The resulting zip archive must have the following structure:

- \ WebHandler \ MyEventProvider.dll
- \ WebRequester \ MyEventResponseHandler.dll
- \ manifest.xml

15.5.6 Installing the template pack

To install the template pack, follow the description in 14.2.3.

15.5.7 Creating a filter rule for opening a alert

After installing the template pack, you will see a new event source named *HTTP GET* in the Event Filter. Right click on the event source and select *Add filter*. Next, right click on the item *New Filter* and select *Edit*. Specify *Create alert* as a filter action and select a destination. Save the filter by clicking the *Save* button.

15.5.8 Testing the template pack

To test the template pack, open the browser and send a HTTP GET request to the SOAP Connector SDK main handler page including the required parameters for raising an event via the *HTTP GET* handler template.

HTTP://localhost/EAWebService/EventConnectorServer.ashx?Handler=HTTP%20Get&action=RaiseEvent&Username=Administrator&Password=&MyParam1=a&Myparam2=b

If all settings have been configured correctly, an alert should be opened in Enterprise Alert® and the status update information should be sent via *HTTP GET* to HTTP://providerhost/EAEventHandler/EAHandler.aspx. To verify it, check the alerts in the Alert Center and the event messages in the Message Center.

16 CONNECTOR TEMPLATE LICENSING

Before you can use your custom templates with Enterprise Alert® you must obtain a valid license for each event provider template. Afterwards, your custom template can be installed and used in Enterprise Alert. This section provides details about how you can obtain a license for the templates you would like to develop.

16.1 Requesting a license

Before you can request a license from the Derdack Sales team, you need to generate a public/private key pair. The private key must be included in your template, while the public key must be sent to the Derdack Sales Team. When the template is loaded by the SOAP Connector SDK, it is verified that the template is the owner of the private key matching the public key sent to Derdack. Here the template is requested to sign some text with its private key. The SOAP Connector SDK then verifies the signature by checking the signed text against the template's public key. If the signature is valid the template will be loaded, otherwise not.

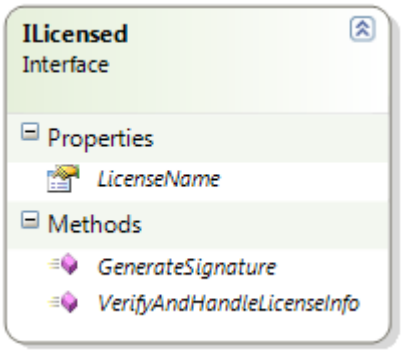
The algorithm for signing is RSA / SHA1. First a SHA1 hash must be build over the given data and then the hash must be encrypted using RSA.

For this reason, you need to generate a public/private key pair for RSA. To do this, you can use the Derdack crypt tool located in the Enterprise Alert® program folder. In addition, you will need to generate a hardware key with the license manager of Enterprise Alert® and then send it together with the public key and a license name to Derdack Sales. The Sales team will respond with a license file. If the license name cannot be used, the Sales team will contact you with a suggestion for a new name. After you have received your license file from Derdack you need to install the license with the license manager of Enterprise Alert®.

The next sections provide more details about a correct implementation of the licensing model described above.

16.2 Implementing licensing in.NET based templates

If you have already developed a server template as a .NET class, you would have already implemented the interface *IEAEventProvider*. In addition to this interface, you will need to implement the interface *ILicensed*.



16.2.1 The property LicenseName

In the property *LicenseName* your template class must return the license name of your template. Using this name, the SOAP Connector SDK can find the matching public key previously sent to Derdack.

```
private const string LICENSE_NAME = "HttpSoap";
public string LicenseName
{
      get { return LICENSE_NAME; }
}
```

16.2.2 Implementation of GenerateSignature()

In the implementation of *GenerateSignature()* your template has to sign the given parameter with its private key by using the RSA / SHA1 algorithm. The return value must be the signature.

In the assembly *Derdack.Common* you will find the *CryptHelper* class for doing this procedure. The following example implementation of *GenerateSignature()* shows how to generate a signature by using *Derdack.Common.Security.CryptHelper*.

```
using Derdack.Common.Security;

private const string PRIVATE_KEY = "BwIAAACkAABSU0EyAA... ";
public string GenerateSignature(string strText)
{
      CryptHelper cryptHelper = new CryptHelper();

      string strSignature = "";
      cryptHelper.SignText(PRIVATE_KEY, strText, out strSignature);

      return strSignature;
}
```

16.2.3 Implementation of VerifyAndHandleLicenseInfo()

VerifyAndHandleLicenseInfo() will be called by the SOAP Connector SDK for providing license information about Enterprise Alert®.

This information can be used in your template to verify the authenticity of Enterprise Alert®. If you don't want to perform authentication, then your method can simply return true.

If you wish to ensure that your template only works with Enterprise Alert®, you should verify the given *EALicenseInfo* object. The following implementation shows how you can do this.

```
private bool m_bEALicenseInfoValid = false;
public bool VerifyAndHandleLicenseInfo(EALicenseInfo licenseInfo)
{
      if (licenseInfo == null)
      {
            // invalid EALicenseInfo object
            return false;
      }

      if (!licenseInfo.IsLicenseInfoValid)
      {
            // licenseInfo should be already marked as valid by the
            // SOAP Connector SDK
            return false;
      }

      if (licenseInfo.EALicensedProvider == null)
      {
            // EALicensedProvider should be exist
            // it stores information about this template license
```


PHONE +49 331 29878-0 · FAX +49 331 29878-22 · WWW.DERDACK.COM
AMTSGERICHT POTSDAM, HRB 13071 · MANAGING DIRECTOR: MATTHES DERDACK · EU VAT ID: DE 812 810 585

```
            return false;
        }

        if (!licenseInfo.EALicensedProvider.Licensed)
        {
            // EALicensedProvider must be licensed
            return false;
        }

        if (licenseInfo.EALicensedProvider.LicenseName != LicenseName)
        {
            // LicenseName of EALicensedProvider must be the
            // license name of this template
            return false;
        }

        // Check the signature to authenticate Enterprise Alert
        CryptHelper cryptHelper = new CryptHelper();
        If (!cryptHelper.VerifySignedMessage(
            CryptHelper.MESSAGEMASTER_SIGNATURE_PUBLIC_KEY,
            licenseInfo.RawLicenseInfoMessage))
        {
            // Signature is invalid. Was it not signed by the private key
            // of Enterprise Alert?
            return false;
        }

        // Set the member m_bEALicenseInfoValid to true.
        // The member will be checked in each request handling method and the
        // value is false by default.
        m_bEALicenseInfoValid = true;
        return true;
}
```

16.3 Implementing licensing in XSL based templates

Licensable XSL templates must provide a signature built by the private key of the template from the whole content of the XSL file.

To do this, you can use the Derdack Cryptography Tool located inside the Enterprise Alert® installation folder. If you already have a private key, then copy it into the text box *Private Key*, otherwise click *Generate public key pair*. Next, copy the content of your XSL template into the text box *Text or XML*. Click *Create Signature*. The text box *Signature* will be filled with the generated signature.

Finally, open your XSL file and add the following XML fragment to it:

```
  <xsl:template match="mm_license">
    <License>
      <LicenseName>
        HttpPost
      </LicenseName>
      <Signature type="Full">
QGfs8wyvbUUKfw4tdYX+PY4EpwsVZNgSbHrBOVxBYDv3IfqdS2AZE9U8K4ZG3pptZpFZ9rQxhvbe
I1HYozpmRJuhGRLJNkChzjI8SrcYLDe1jbMVlO5d/1qUZ2OoDewx7QB37kHNVM23Gug9OGU5XKrt
2yPgJEl+OjfVEEQGLJg=
```

```
      </Signature>
    </License>
  </xsl:template>
```

Every time the template is loaded into the SOAP Connector SDK, the signature will be verified.

Please do not modify the content of the XSL stylesheet file after you have added the License element, otherwise the signature will become invalid.

17 FURTHER INFORMATION

Please feel free to contact Derdack for further information or consultancy:

Derdack GmbH
Friedrich-Ebert-Strasse 8
14467 Potsdam
Germany

Phone: +49 (0) 331 29878-0
Fax: +49 (0) 331 29878-22

Mail: support@derdack.com
www.derdack.com