EnterpriseAlert® 2017

Application Programming Toolkit (APT), Getting started

1 OVERVIEW

In addition to the standard alerting workflows which available in each Alert Policy, Enterprise Alert® comes with an Application Programming Toolkit to perform custom notification and alerting operations upon receipt of events. The Application Programming Toolkit offers far more flexibility and customization opportunities for notification and alert processes but requires some developing skills and know-how.

Enterprise Alert® receives incoming events from 3rd party interfaces (e.g. a command line interface or a System Center Connector). Such events are then forwarded to the internal **Alert Policy system**. The system evaluates the received data and triggers a specific alert policy. Besides the standard alert and notification options, an Alert Policy can also be configured to execute your custom alert and notification script.

A script must be written in a supported scripting language (e.g. Java Script or VB Script). The Scripting Host uses the **Microsoft Windows Scripting Host** and scripting engines (by default the JavaScript engine and the VBScript engine) for interpreting and executing scripts.

The Scripting Host provides functions for creating, modifying and sending notifications (e.g. SMS) through the Notification Channels of Enterprise Alert®. The Scripting Host fires a matching event based on the event type when receiving a new event through the Alert Policy system. An event handler can be implemented in a script to handle the matching event.

In addition to basic event management functions, the Scripting Host provides functions for creating and modifying Alerts and events for monitoring the status of an alert during processing.

Beyond this, you can also execute scripts as Remote Actions. This enables you to remotely trigger scripts from mobile applications or various media such as SMS or email. This would be useful for example if you would like to ping a server, while not being at your desk, amongst many other applications.

For complex activities (e.g. database access) you can simply use external **ActiveX** components in the script.

Other features of the Scripting Host are:
- Full Unicode support and Unicode auto detection as well as conversion into outgoing messages.
- Cloning and parallel execution of more than one instance of a script to increase performance.
- Logging mechanism to debug script applications.
- Web-based user interface to:
  - o edit script application settings
  - o upload, download and edit script code
  - o send test messages
  - o view and delete script log files

The Scripting Host is part of the default Enterprise Alert® installation and runs as an external Windows service application. For communication between the Scripting Host and Enterprise Alert, **Microsoft Message Queue (MSMQ)** is used. The Scripting Host needs to be licensed for any scripts to run.

> Before creating script applications in Enterprise Alert, check whether both the Scripting Host and Enterprise Alert are running and are properly licensed. Microsoft Windows Scripting Host needs to be installed as well.

2 USING THIS DOCUMENT

This document was initially written for an older version of Enterprise Alert®. It contains code samples and file names that might contain an old terminology. The table below lists deprecated entity names and lists corresponding new expressions that are valid with Enterprise Alert® 2017.

Deprecated entity name	Translation for Enterprise Alert® 2017
Ticket	Alert
Escalation Chain	Team (Escalation within that team can be done with Team Escalation notification type. Please refer to section 5.8 for further details)
Group	Team (Group Alerts to all team members simultaneously can be done with Team Broadcast notification type. Please refer to section 5.8 for further details)
Filter	Alert Policy

3 CONFIGURATION OF ENTERPRISE ALERT®

Part of the main functionality of the Scripting Host is the handling of messages that are received by Enterprise Alert®. In your scripts you determine what should happen if an incoming message of a certain type triggers your script. Possible actions for your script are described in [Writing Scripts](.).

The Scripting Host is included in the default installation of Enterprise Alert. It is important to note that Enterprise Alert does not automatically recognize whether messages should be forwarded to the scripting host. Therefore, in order to forward messages to the scripting host, you need to create at least one Alert Policy inside the web portal of Enterprise Alert® that forwards the events or messages onto a specific script running under the scripting host. In the Alert Policy, you will then need to select the "Custom (APT)" as Alert Type, where you can then choose which script to direct the message to.

You can also determine what policy conditions should apply to the event or message that Enterprise Alert® receives. One condition could be, for instance, to forward only those messages to your script that have a specific severity such as *critical*. Another example of a condition could be to forward only the events/messages with the text that contains the keyword "script".

3.1 Alert Policy Setup

In Enterprise Alert® events can be received from various event sources (3rd party interfaces). Related to each of these sources you can create your Alert Policies that match to events which Enterprise Alert® receives through the source.

This includes the **SCOM Connector**, the SCO Connector, the SCSM Connector, the ITM Connector, the HPOM Connector **Command Line** and all available **2-way communication media** like SMS, MMS, E-mail, Voice and Instant Messaging.

3.2 Alert Policy Setup

In the web portal of Enterprise Alert® navigate to the Alert Policies overview by clicking *Policies->Alert Policies*. Click *Create New* on the bottom right and create a new Alert Policy for events coming in through your preferred event source. On the tab *Alerting*, click *Custom (Script)*:

Select the script application that you have created using the guidelines of the following sections under *Script for custom notification workflow:*

4 WORKING WITH THE SCRIPTING HOST

The Scripting Host and the script applications hosted by it are configured through the web portal of Enterprise Alert. To do this, open the web portal and from the menu select System -> Scripting Host. The Scripting Host configuration page is displayed. A list of all the available scripts is displayed here. Underneath this list settings for the general configuration of all script applications are displayed.

4.1 General Configuration

Before adding or editing a script application, please take a look at the general configuration of the Scripting Host:

- **Convert attachments on sending**: This property is important only for sending messages with attachments (e.g. MMS or E-mail) from a script to Enterprise Alert®. If the property is set to **2 - To Mime**, the attachments will be converted into mime format first, before the message is sent to Enterprise Alert®. **1 - To Path** stores the mime attachments as files. The paths will be set in the message, before it is sent to Enterprise Alert®. For a detailed description on how to handle messages with attachments, please see [Handling attachments](#)

- **File Logging**: You can find this option under the path System -> General -> File Logging and set it by selecting the Logging Level for Scripting Host. If this option is enabled, the activities of Scripting Host will be logged in the file **ScriptingHost.log**. This log file is stored in the working folder of the Scripting Host. If you do not want to log your files, we recommend the deactivation of this option. A lot of data will be written into the log file, and the file can become quite large after a short period.

4.2 Add or Edit a Script Application

Click the **Edit** button at the end of a list entry to change the configuration of an existing script application or click the **Create New** button if you want to create a new one. In both cases the following settings are available:

© Derdack GmbH | Derdack Corp., www.derdack.com
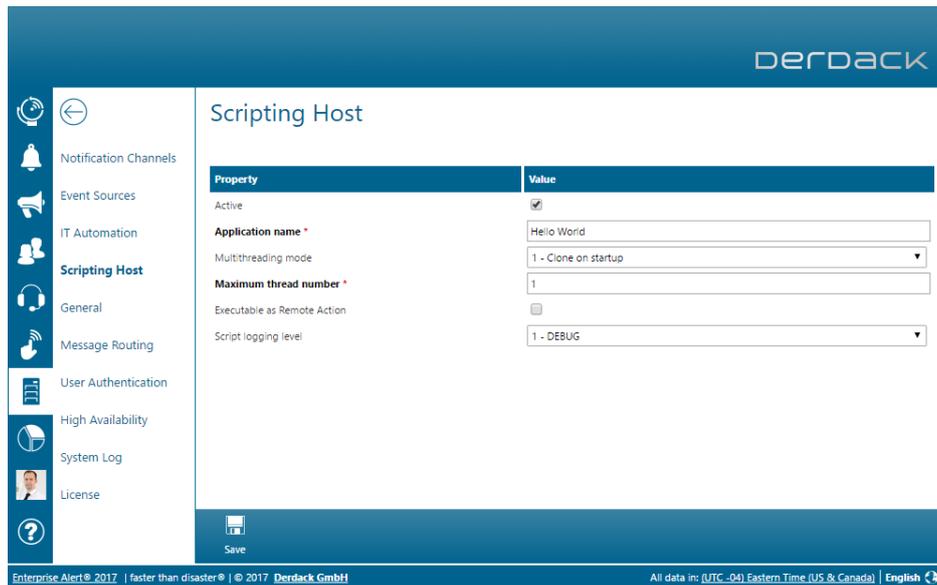
The configuration page shows various properties to be set. Mandatory fields are always marked bold and with an asterisk (*). The following properties have to be configured for creating a fully operational script application:

- **Active**: Activates or deactivates the application
- **Application name**: A unique name for the script application
- **Multithreading mode**: Scripting Host supports three different threading modes for the parallel execution of a script:
  - ***(0) Off***. Only one script clone handles all the request messages. This mode is similar to **Clone on startup**, but with a **Maximum thread number = 1**.
  - ***(1) Clone on startup***. The scripting engine will be cloned and started parallel on *n* instances immediately when the application starts. Thus, **n** instances of the engine will permanently wait for incoming messages to process them. The value **n** represents the configured value in the **Maximum thread number** property below. In this threading mode the script application is able to respond faster than in the second option **Clone on invoke**. However if there is no message to handle, **n** instances of the engine still have been cloned and loaded in memory.
  - ***(2) Clone on invoke***. The scripting engine will be cloned in case the script application is requested by an incoming message. If another request calls the application and the Maximum thread number is not reached, the engine will be recloned.
- **Maximum thread number**: The maximum number of script clones (please see **Multi threading mode**).
- **Script logging level**: Only log entries with an equal or higher logging level as this value will be added to the log file. The level order is as follows: **Debug** (lowest), **Info**, **Warning**, **Error**, and **Fatal** (highest). The value **Off** deactivates logging.

Once you have made the application settings, press **Save**.

The **Status** property displays the application status information (e.g. **OK, Error, No Script code** etc.) and is available when editing the script application or in the **Script applications** list.

© Derdack GmbH | Derdack Corp., www.derdack.com

4.3 Edit the Script Code

Once you have added a new active script application or clicked the Script code Edit button in the script applications list, the following page opens:
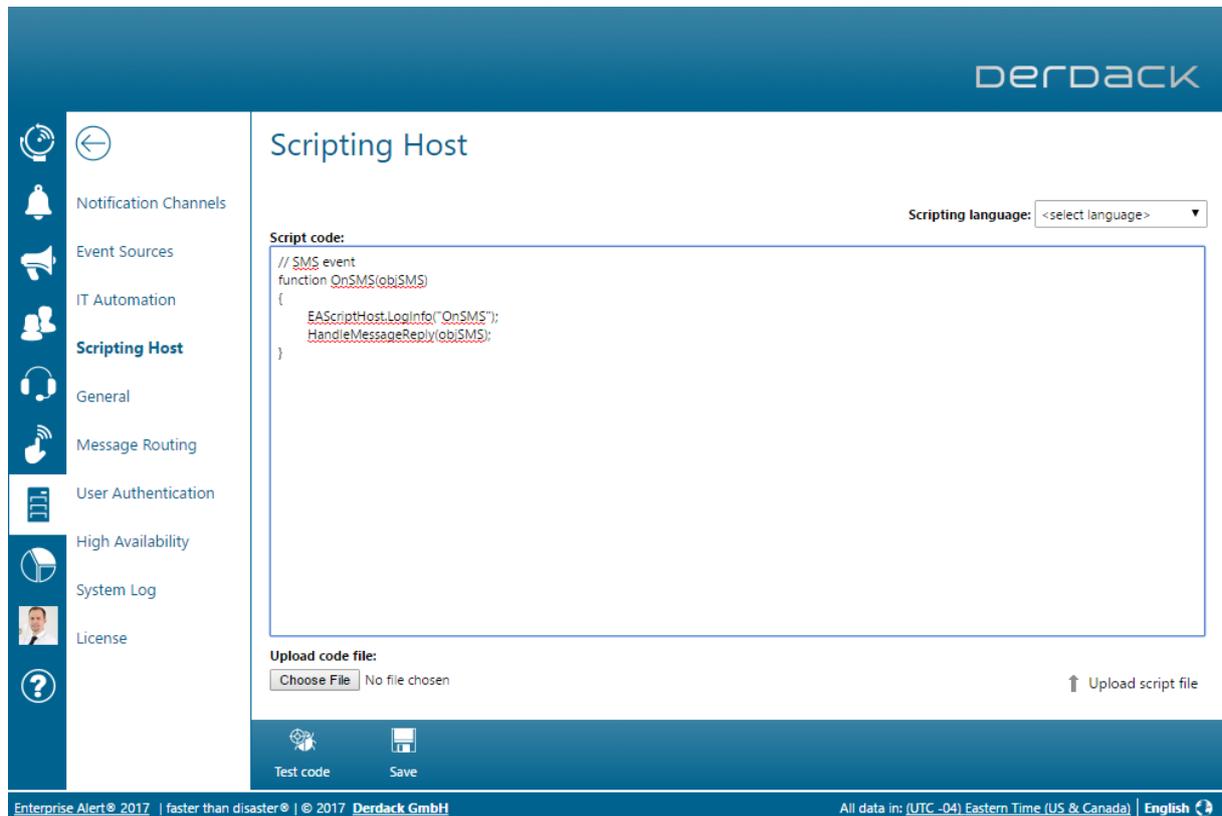
Use the **Script language** drop down list to adjust the script language to your entered code.

Script code: Here you can edit the script code. Please see Writing Scripts on how to write scripts for the Scripting Host.

To update the script code from an existing script file, first browse and select a script file to upload. Then select **Upload script file** to upload the file. The script language will automatically be set to the file extension of the script file (e.g. .js, .vbs).

Press the **Save** button. The script code will be parsed in the script application. If there are syntax errors in the code, an error message will be displayed and the code will not be set in the application. Please note though that if the script application is not active, the code will not be checked for syntax errors. If there are syntax errors, then the script will not load if the script application is then activated. The script would then have to be corrected and resaved.

If the code is correct it will be inserted into the application and the application will be restarted. Click **Test code** to open the page, where a test message can be sent to your script application.

4.4 Sending a Test Message to the Script

The field **Destination service** represents your script. **Origination service** shows the originating service of the original message. Enter or change the value in the **Destination** field if required, depending on the conditions in your script. Enter some message text and click **Send**. The message will now be transmitted to the Scripting Host, and the corresponding implemented handler function for the **origination service** e.g. **OnSMS()** will be called in your script application.

You can check whether the handler function worked correctly by viewing the log files of your application. To view the logs, click the corresponding **logs** button under the **Script applications** list. The following list of log file(s) will be displayed:

Here you delete log files by ticking the respective check box next to the log file name and by pressing the red **Delete** button. Click the **View log** button to download the content of the log file and to display it in your text editor.

4.5 Triggering a Script via a Remote Action

In order to be able to trigger a script via a Remote Action, you first need to enable this in the script's properties:
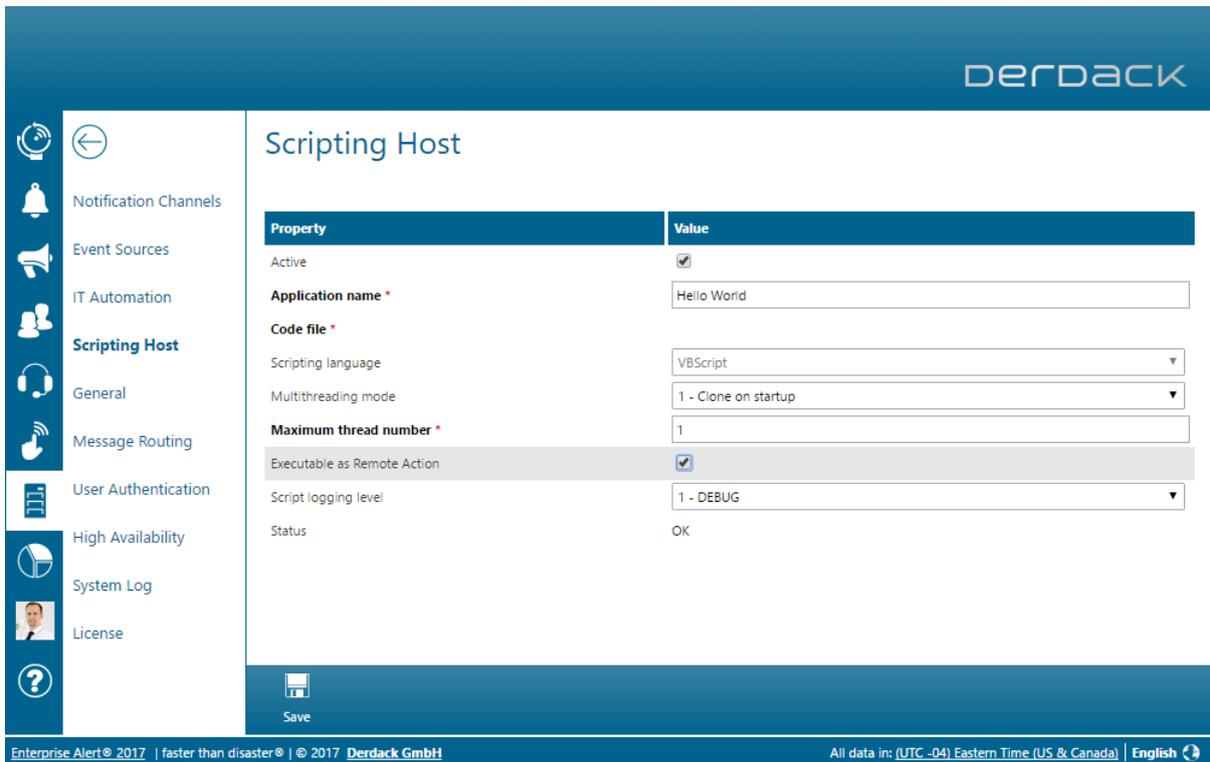
Once the script can be executed as a Remote Action, you will need to finally create a Remote Action to actually trigger the script. First, you create the Remote Action as usual, except that under **General, Execute In** needs to be set to Scripting Host:
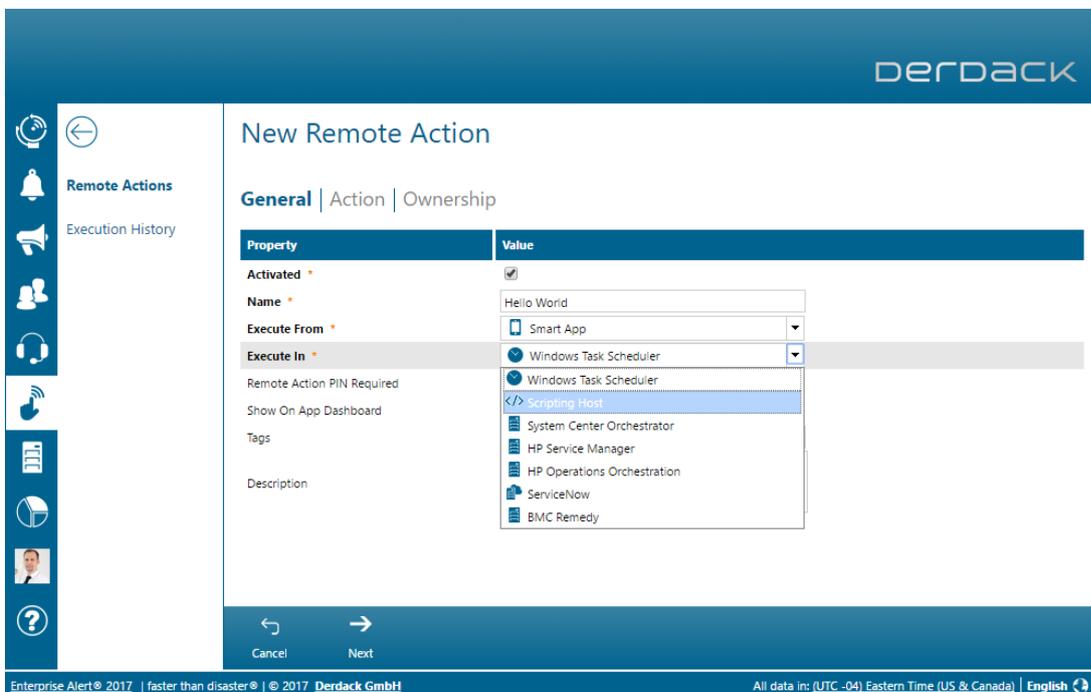
Next, all functions available in the script that can be executed will be listed. The Remote Action only executes a single function in the script, so you will need to select the script that you would like called by the Remote Action:
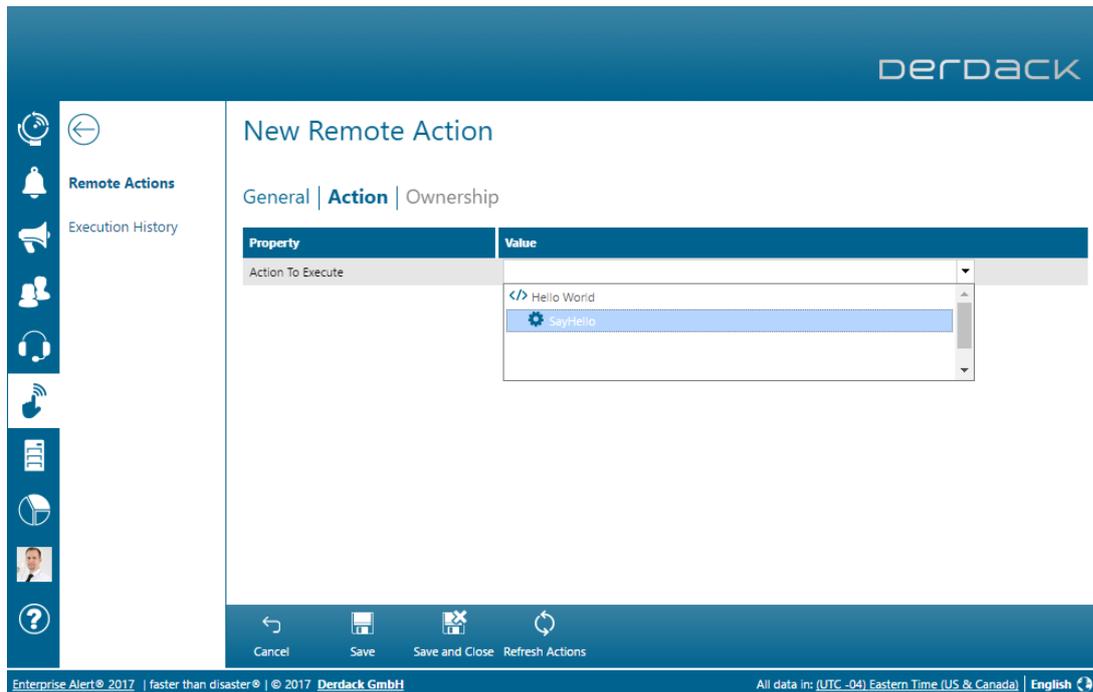
Once you have select the function, all available parameters that the function has will be listed. The user can input these parameters before executing the Remote Action. These parameter values are then passed onto the function by the Remote Action and the function executes as usual.

It is also possible for the script to provide feedback to the user after the function in the script has finished execution. This way, the user can for example see whether the script was able to successfully ping the target server as well as what exactly was output by the ping. An example script can be found under Executing from Remote Actions.

5 WRITING SCRIPTS

The following sections provide a detailed description of how scripts can be written for use within the Scripting Host of Enterprise Alert®. The following section contains functions and how they can be used by presenting example script code extracts written in JavaScript.

> Please note that you be should familiar with script writing to create scripts to be executed by the Windows Scripting host.

The object that exposes the main functions of the Scripting Host is **EAScriptHost**. This object can be used to create message objects, manage tickets, create ActiveX components and write log entries for debug purposes.

The base message object is **objMsg**. An instance of this object can be created by EAScriptHost with the function **CreateSMS()** etc. The type of object returned depends on the function called to create the object e.g. **CreateSMS()** returns a **objSMS** object, while **CreateMMS()** returns a **objMMS** object. The **objMMS** object has additional functions available for managing attachments. A message object can also be passed as a parameter to a message handling function. Please refer to the sections below for a further description of how the functionality of these objects are used, otherwise a complete reference is found under Object and function reference.

5.1 Logging

To write log entries in the log file of your script application in order to debug your script code, you can use the following functions of the **EAScriptHost** object:

- void EAScriptHost.LogDebug(strText)
- void EAScriptHost.LogInfo (strText)
- void EAScriptHost.LogWarning (strText)
- void EAScriptHost.LogError (strText)

The functions write the value of **strText** to the log file depending on the configured **logging level** of your script application.

For example: To write a log entry with the text "Hello World!" and the logging level **Info** enter the following line in your script:

```
EAScriptHost.LogInfo("Hello World!");
```

The logging of this example will not work if the logging level of your application is higher than Info (so Warning or Error). You can for instance enter a lot of LogDebug() entries in your script. If you have successfully tested your application, you can switch the application log level to the higher value Warning to log only warnings and errors and make your log file smaller and clearer.

5.2 Message Types

Each message sent to or received from Enterprise Alert® takes the form of xml. This xml is encapsulated by the **objMsg** object, which is available through the Scripting Host. All other message type objects inherit from this object.

The following message types are supported by Enterprise Alert: SMS, MMS, Email, Instant Messaging, Paging, Voice Call, Fax, FF and FM. FF stands for "Find Me Follow Me", while FM stands for "Favorite Media".

Each message type has a corresponding object which is available through the Scripting Host: **objSMS**, **objMMS**, **objEmail**, **objIM**, **objPaging**, **objVoiceCall**, **objFax**, **objFF** and **objFM**.

Additionally there is a special message type for creating tickets, namely **objTicket**.

5.3 Create Message Objects

To create a new message object, call the corresponding function on **EAScriptHost**: **CreateSMS(), CreateMMS(), CreateEmail(), CreateIM(), CreateVoiceCall(), CreatePaging(), CreateFax(), CreateFF()** and **CreateFM()**. The return value of these functions is an instance of the corresponding message object.

Example: To create a new **objSMS** object**,** enter the following lines in your script:

```
var objSMS = EAScriptHost.CreateSMS();
if (objSMS == null)
{
    EAScriptHost.LogError("Could not create message object");
```

```
}
```

5.4 Get and Set Properties of Message Objects

In order to set and get properties of message objects you have to call the functions **GetProperty**() and **SetProperty**() of **objMsg**. On **GetProperty**() you enter the property name in the parameter. The property value will be returned. On **SetProperty**() you enter the property name in the first parameter and the value in the second parameter. The return value is true if the **objMsg** supports this property, or false if not.

objMsg supports the following properties

- **mm_body**: the message text
- **to**: the recipient mobile number of the message
- **from**: the originator mobile number of the message
- **serviceTo**: the service that should send the message (e.g. "/SMPP/service1")
- **serviceFrom**: the originator service (generally "/ScriptingHost/<ScriptAppName>")
- **options**: the message options (see section supported message options)
- **index**: the index of the message (will be overwritten by Enterprise Alert ® on send)
- **type**: the type of the message (should be set to "user")
- **subtype**: the sub type of the message (should be set to "outbound")
- **timestamp**: the date and time that the message was created. Warning: if this property is set, then it needs to be in the correct format, otherwise Enterprise Alert will not process the message. The correct format is yyyy-mm-dd hh:mm:ss
- **priority**: the message priority. This can be one of the following values:
  - o 0: low priority
  - o 1: major priority
  - o 2: critical priority

Other message types inheriting from **objMsg**, support properties that are specific to the type of message.

- **subject**: the subject of the message (supported by **objMMS** and **objEmail**)
- **externalTicketId**: a custom id assigned to a ticket e.g. for a 3rd party application (supported by **objTicket**)

The script below demonstrates setting message properties:

```
var objSMS;
objSMS = EAScript.CreateSMS();
if (objSMS != null)
{

    objSMS.SetProperty("mm_body", "This is a test SMS");
    objSMS.SetProperty("to", "123456");
    objSMS.SetProperty("from", "654321");
    objSMS.SetProperty("serviceFrom", "/Script Name");
    objSMS.SetProperty("serviceTo", "/Routing/SMS");
    objSMS.SetProperty("options", "0"); // Default messaging option
    objSMS.SetProperty("timestamp", " 2009-11-07 13:42:31");
    objSMS.SetProperty("priority", "0"); // Low message priority
    if (objSMS.Send())
                EAScriptHost.LogDebug("Message sent successfully");
    else
                EAScriptHost.LogError("Message could not be sent successfully");

}
else
```

```
{
    EAScriptHost.LogError("SMS object could not be created");
}
```

5.5 Handling Incoming Messages

The Scripting Host contains 5 functions for the handling of incoming messages: **OnSMS()**, **OnMMS()**, **OnEmail()**, **OnVoiceCall()**, **OnIM()** and **OnNewEvent()**. All that needs to be done to handle an incoming message is to provide a function with the name of the corresponding event. Please see [Message events](#) for further information.

```
function OnSMS(objSMS)
{
    EAScriptHost.LogInfo("SMS received");
}
```

5.6 Generate Answer Messages and Return Them to the Originator

To create an answer of an incoming message you have to generate a new message and set the properties **to** and **serviceTo** with the values of the **from** and **serviceFrom** properties of the incoming message. To simplify this procedure, **objMsg** provides the function **CreateAnswer()**. This function will return an answer message of the original message with an empty message body (property **mm_body**).

To send a message, **objMsg** provides the function **Send()**. This function returns true if the message is syntactically correct, otherwise false. The function transmits the message to the service specified in the **serviceTo** property. You can check in the web portal of Enterprise Alert under Messages --> Message Center to determine whether the message was successfully sent.

Example: The handler function in the following example creates an answer of the incoming message, sets the message text to "Reply to <incoming message text>" and sends the message back to the originator.

```
function OnSMS(objMsg)
{

    // get the message text
    strMsgText = objMsg.GetProperty("mm_body");
    EAScriptHost.LogInfo("Receive msg with text: " + strMsgText);

    // create answer
    objAnswer = objMsg.CreateAnswer();

    // set the message text
    objAnswer.SetProperty("mm_body", "Reply to " + strMsgText);

    // send the message back
    if (objAnswer.Send())
    {
            EAScriptHost.LogInfo("Message sent OK !");
    }
    else
    {
            EAScriptHost.LogError("Could not send message !");
    }
}
```

5.7 User and Address Functions

EAScriptHost.IsUserMemberOfAccount() can be used to determine if a user account belongs to the specified user group, escalation chain or schedule. This would be useful for example if a message has been received, but only users belonging to a specific group must get responses to the received message.

Typically, messages received from Enterprise Alert contain the raw address of the sender. **EAScriptHost.IsUserMemberOfAccount()** requires that a username be passed as a parameter. Therefore the function **EAScriptHost.GetProfileNameByAddress()** would be used to retrieve the profile name for the specified address.

Please see boolean IsUserMemberOfAccount(strUser, strAccount) and string GetProfileNameByAddress(strAddress) for more information.

5.8 Ticket Management Functions

5.8.1 Creating, Retrieving and Deleting a Ticket

To create a ticket, use **EAScriptHost.CreateTicket()** and then **objTicket.Send()** to send the ticket to Enterprise Alert for processing. If processing is successful, then Enterprise Alert creates the ticket and starts ticket processing.

```
// Set the external id here, can be arbitrary
var strExternalId = "";

var ticketObj;
ticketObj = EAScriptHost.TicketCreate("TestUser", "/Routing/SMS", 0, "A ticket has been initiated for event " + strExternalId);
if (ticketObj == null)
{
    EAScriptHost.LogError("Ticket not created");
}
objTicket.SetProperty("externalTicketId", strExternalId);

// Set the notification type to User alert
objTicket.SetProperty("notificationType", "1");

// Now send the ticket to Enterprise Alert for processing
if (ticketObj.Send() == false)
{
    EAScriptHost.LogError("Ticket not dispatched to Enterprise Alert");
}
```

The alert destination and the desired notification type can be set explicitly with use of the property **notificationType. It can be set to one of the following values:**

- 0 = Automatic
- 1 = User alert
- 2 = Team Broadcast
- 3 = Team Broadcast to a schedule
- 4 = Team Escalation

- 5 = Team Escalation to a schedule
- 6 = Subscription Feed

If the value of the property is zero or not even specified, the destination type will be determined automatically. If the destination was determined as a Team, the alert will be processed as Team escalation by default.

To retrieve a ticket, you can call one the following functions. **EAScriptHost.TicketGetByUniqueId()**, **EAScriptHost.TicketGetByExternalId()** and **EAScriptHost.TicketGetByLastInternalId()**. The function that you use will depend on your scenario.

The ticket's unique id is the unique id of the ticket's record in the Enterprise Alert database. The unique id is available on the **objTicket** object and can be retrieved using **objTicket.GetUniqueTicketId ()**, or it is passed as a parameter to the **OnTicketStatus()**, **OnEscalationNextMember()** or **OnMemberNextMedia()** events. Therefore, one would use **EAScriptHost.TicketGetByUniqueId()** when one has access to the unique id of the ticket.

The external id of the ticket can be assigned as an arbitrary/custom id when creating the ticket. You can set the external id using **objTicket.SetProperty("externalId", value)** for example.

The internal id is the ticket id that is displayed in all ticket messages sent out. It is an easy to remember and reasonably short number that is automatically generated by the system on ticket creation. Because these numbers may cycle (i.e. if the number of the internal id reaches a certain limit, the internal id starts at 1 again), there may be more than 1 internal id for the same number available. **EAScriptHost.TicketGetByLastInternalId()** retrieves the most recently created ticket with the given internal id. As mentioned, the internal id is available in all ticket messages sent out and can therefore be extracted from the message text.

To delete a ticket, call **EAScriptHost.TicketDelete()** passing the ticket's unique id as a parameter.

```
function OnTicketStatus(nTicketStatus, nUniqueTicketId, strInternalTicketid, strExternalTicketId)
{

    // Get the ticket using the ticket's unique id
    var objTicket;
    objTicket = EAScriptHost.TicketGetByUniqueId(nUniqueTicketId);
    if (objTicket == null)
    {
        EAScriptHost.LogError("Ticket could not be retrieved");
    }

    // Delete the ticket if it has the specified external ticket id
    if (strExternalTicketId == "PleaseDeleteMe")
    {
        if (EAScriptHost.TicketDelete(nUniqueTicketId) == false)
        {
            EAScriptHost.LogError("Ticket could not be deleted successfully");
        }
    }

}
```

5.8.2 Monitoring a Ticket's Status

Whenever the state of a ticket changes, the **OnTicketStatus()** event is fired. In this way the status of the ticket can be tracked as it changes. There are 2 additional events which may be of use: **OnEscalationNextMember()** and **OnMemberNextMedia()**. These events are fired as the next member or media in an escalation chain is processed. Bear in mind that all of these events are fired asynchronously. What this means is that the ticket status as retrieved from the database may not match the ticket status passed as a parameter to the **OnTicketStatus()** event.

```
function OnTicketStatus(nTicketStatus, nUniqueTicketId, strInternalTicketid, strExternalTicketId)
{
    EAScriptHost.LogInfo("Ticket status changed");
}

function OnEscalationNextMember(strMemberName, nUniqueTicketId)
{
    EAScriptHost.LogInfo("Next member in escalation chain processed");
}

function OnMemberNextMedia(strUserName, strMediaType, nUniqueTicketId)
{
    EAScriptHost.LogInfo("Next media type for the specified user account being processed");
}
```

5.8.3 Creating and Managing Messages for User Accounts Associated with a Ticket

All of the ticket messages associated with a ticket at a given point in time can be queried and accessed using **objTicket.GetTicketMessageAt()** and **objTicket.GetTicketmessagesCount()**. Using the **objTicketMessage** object that is returned from **objTicket.GetTicketMessageAt()**, you can create new messages or simply query the ticket message details.

When a ticket is acknowledged, resolved or if any of the ticket message information changes, then the ticket will have to be re-retrieved. See [Creating, retrieving and deleting a ticket](#) for information on retrieving a ticket.

```
// Refresh the ticket details
var objTicket = EAScriptHost.TicketGetByUniqueId(nUniqueTicketId);
var nMsgCount = objTicket.GetTicketMessagesCount();

for (var i=0; i<nMsgCount; i++)
{

    var objTicketMsg = objTicket.GetTicketMessageAt(i);
    var nMsgStatus = objTicketMsg.GetTicketMessageStatus();
    var strTicketMsgUser = objTicketMsg.GetUserName();

    // Only dispatch messages to recipients to whom a message has been submitted successfully
    if ( (nMsgStatus == objTicketMsg.TicketMessageStatusDelivered || nMsgStatus == objTicketMsg.TicketMessageStatusBuffered ||
nMsgStatus == objTicketMsg.TicketMessageStatusAnswered || nMsgStatus == objTicketMsg.TicketMessageStatusNoReply)
    {

        var objNewMsg = objTicketMsg.CreateNewMessage();
        objNewMsg.SetProperty("mm_body", "This is an additional message to be sent to all recipients associated with a ticket.");
        objNewMsg.Send();

    }

}
```

5.9 Handling Attachments

Basically Enterprise Alert® and the Scripting Host support two formats of attachments in messages:

- (1) The **mime format** contains the attachments in mime. Binary attachments are encoded in base64 inside the mime.
- (2) The **path format** specifies in the message the path to an external mime file or the paths to content files separated by semicolons. All multimedia files that are supported by MMS messages (e.g. jpeg, gif, smil, etc.) can be used as content files. Relative paths to the globally configured attachment directory, absolute path to files or directories of the local system or URLs are possible.

> If the Scripting Host is running on another host than Enterprise Alert®, the attachments should be in mime format. If the attachments are in path format, the Scripting Host normally cannot access the attachment directory of Enterprise Alert® and vice versa. In this case, set the global option **Convert attachments on sending** to **2 - To Mime**, and Scripting Host will always convert path attachments to mime format on sending a message to Enterprise Alert®. If the Scripting Host is running on the same host as Enterprise Alert®, the attachments should be transferred as paths: the content files don't have to be transferred completely and performance may therefore be increased. In this case the global option **Convert attachments on sending** is set to **1 - To Path**.

The Scripting Host provides functions for setting, modifying and saving attachments to files. These functions are provided by objMsgAttachment, which inherits from **objMsg**. Objects that inherit from **objMsgAttachment** are **objMMS**, **objEmail** and **objTicket**. **objMsgAttachment** has the following functions:

- bool ClearAttachments()
- int GetNumberOfAttachments()
- objMsgAttachment GetAttachment(int nIndex)
- bool RemoveAttachment(int nIndex)
- int AddAttachmentFiles(String strPaths)
- int AddAttachment(objMsgAttachment objNewAtt)
- String SaveAttachments(String strDir)
- String SaveAttachment(String strDir, int nIndex)
- bool SaveAttachmentsToMimeFile(String strDir, String strFileName)
- bool GenerateSmil()
- void SetSmilAutocreation(bool bActivate)

To get the number of message attachments, use the function **GetNumberOfAttachments()**. In order to clear all the message attachments, use the function **ClearAttachments()**. The return value is true if the function has successfully been executed, otherwise false.

To get an attachment specified by its index, use the function **GetAttachment()**. Use a value of the interval from zero to the number of attachments - 1 as parameter. If the parameter value is outside the interval, the function returns **null**. If the function is successful, the attachment on the given index position will be returned as an object of type objAttachment. This object encapsulates the properties of the attachment. To get the name, mime type and file size, use the following functions of **objAttachment**:

- String GetName()
- String GetType()
- int GetFileSize()

Use the function **RemoveAttachment()** of **objMsgAttachment** to remove an attachment. Use the index of the attachment that should be deleted, as a parameter. The function returns true if the function was successful and false if the given index is invalid. To add an attachment of type **objAttachment** you can use the function **AddAttachment()**. The return value is the index position of the newly added attachment in the message or -1 if the attachment could not be added (e.g. the given object is not a type of **objAttachment**). To add attachment(s) from a file(s) you can use the function **AddAttachmentFiles()**. Specify semicolon separated paths to files on the local system and/or URLs to files on accessible web servers as a parameter. The return value is -1 if the attachments could not be added (e.g. the given paths are not valid). It returns the index of the first added attachment if the function call is successful. In case not all attachments could be loaded from the given paths, a warning message will be written into the log file.

In order to save an attachment of a message to a file, use the function **SaveAttachment()**. Specify a valid directory of the local system where the file should be stored in as the first parameter of this function. The second parameter should be the index of the attachment to save. The return value is the complete path of the stored attachment file if successful or null if the directory or the index was invalid.

In order to save all attachments of a message to a directory use the function **SaveAttachments()**. Specify a valid directory of the local system where the files should be stored in as a parameter. If the function is successful, the return value shows the complete paths of the stored attachment files as semicolon separated string or null, in case the directory or the index was invalid.

To save all attachments of a message to a MIME file, use the function **SaveAttachmentsToMimeFile()**. Specify a valid directory of the local system where the file should be stored in as the first parameter. The second parameter is a file name for the target file. The return value is true if successful or false in case the directory or the file name was invalid.

The correct display of multimedia attachments on mobile phones is controlled by a SMIL attachment. In order to generate and to add a SMIL part to a message, use the function **GenerateSmil()**. The function returns true if successful, otherwise false. All existing multimedia attachments will be used for the SMIL generation.

> If you add an attachment after the SMIL part was generated, the attachment will be received by the mobile phone, but may not be displayed properly.

To generate and add a SMIL part automatically before sending an MMS message, use the function **SetSmilAutocreation()** with the parameter true. If you want to deactivate this feature, call the function with parameter false.

The following example script demonstrates the handling of attachments for an incoming MMS message.

```
// handler function for incoming MMS messages
function OnMMS(objMsg)
{
    // create an answer MMS message (attachments will not be copied)
    objAnswer = objMsg.CreateAnswer();
    objAnswer.SetProperty("subject", "mms answer");

    // get and log number of attachments
    nAttCount = objMsg.GetNumberOfAttachments();
    EAScriptHost.LogInfo("Number of attachments : " + nAttCount);
```

© Derdack GmbH | Derdack Corp., www.derdack.com

```
// iterate all attachments, log the properties
for (nIndex = 0; nIndex < nAttCount; nIndex++)
{
          objAtt = objMsg.GetAttachment(nIndex);
          strAtt = "Attachment " + nIndex + ": ";
          strAtt += "Name=" + objAtt.GetName() + ", ";
          strAtt += "Type=" + objAtt.GetType() + ", ";
          strAtt += "FileSize=" + objAtt.GetFileSize();
          EAScriptHost.LogInfo(strAtt);

          // save non jpg image attachments to files
          if (objAtt.GetType() != "image/jpg")
          {
                    objAnswer.SaveAttachment("C:\\test", nIndex);
          }
          // add big jpg images (> 1kb) to answer message
          else if (objAtt.GetFileSize() > 1000)
          {
                    objAnswer.AddAttachment(objAtt);
          }
}

// remove the first attachment
if (! objMsg.RemoveAttachment(0))
{
          EAScriptHost.LogError("Could not remove first attachment !");
}

// add attachments
if (objMsg.AddAttachmentFiles("c:\\1.gif;http:/host/2.jpg") < 0)
{
          EAScriptHost.LogError("Could not add attachments !");
}

// generate SMIL (overwrites existing one)
if (! objMsg.GenerateSmil())
{
          EAScriptHost.LogError("Could not generate SMIL !");
}

// save attachments to files
strPaths = objMsg.SaveAttachments("C:\\test");

if (strPaths == "")
{
          EAScriptHost.LogError("Could not save attachments !");
}
else
{
          EAScriptHost.LogInfo("save attachments :" + strPaths);
}

// save answer attachments to a MIME file
objAnswer.SaveAttachmentsToMimeFile("C:\\test\\", "test.mime"));

// activate SMIL creation on sending
objAnswer.SetSmilAutocreation(true);

// send answer message to Enterprise Alert®
objAnswer.Send();
}
```

© Derdack GmbH | Derdack Corp., www.derdack.com

5.10 Using External Active Components

To create an instance of an external ActiveX component, **EAScriptHost** provides the function **CreateActiveX().** As parameter enter the **ProgID** of the ActiveX component to be created. If the ProgID is correct, the return value is an object of the required component type, otherwise null.

```
// Create the ActiveX object
var msword = EAScriptHost.CreateActiveX("Word.Basic");
if (msword != null)
{
    // Show the word application
    msword.appshow();
    // Open a new file in the word application
    msword.filenew();
    // Add some text to the word document, can be a message that has just been received.
    msword.Insert("text");
}
```

5.11 Performing Database Operations

The following script demonstrates performing basic operations with a database i.e. connecting to the database, retrieving records from the database and updating a record in the database.

```
// Connect to the database
var oConn;
oConn = new ActiveXObject("ADODB.Connection");
oConn.Open("Driver={SQL Server};Server={Your Server};Database={Db};User Id={User};Password={Password};","","");

// Retrieve data from the database
var strResult = "";
var strLastRowValue = "";
var oRs = oConn.Execute("SELECT * FROM MyTable");
while (!oRs.EOF)
{
    strResult = strResult + oRs.Fields.Item("MyColumn").Value;
    strLastRowValue = oRs.Fields.Item("MyColumn").Value;
    oRs.MoveNext();
}
oRs.Close();

// Update the last row
oConn.Execute("UPDATE MyTable SET MyColumn = 'last row' WHERE MyColumn = '" + strLastRowValue + "'");
oConn.Close();
oRes = null;
oConn = null;
```

5.12 Executing from Remote Actions

Any function defined the script can be called by the Remote Action. Any parameters that you define for the function will become automatically available when defining the Remote Action. The user can then input values via the Mobile App, which are then passed in via the function's parameters when the function is called. Important to note is that every executed Remote Action requires a result in order for the Remote Action to complete execution. This result needs to be manually set at the end of the function via **RAContext.SetExecutionResult()**.

The following script demonstrates how you can use a script to ping a target server via a Remote Action:

© Derdack GmbH | Derdack Corp., www.derdack.com

```
function Ping(serverOrIP) {

    EAScriptHost.LogInfo("Sending ping to: " + serverOrIP);
    var strParams = serverOrIP;

    try {

        // Execute the ping. The result for the RemoteAction will be set in ExecuteCommandWithResults
        var execCode = ExecuteCommandWithResult("Ping", strParams);
        EAScriptHost.LogInfo("Command successfully executed.");
    }
    catch (e) {
        EAScriptHost.LogError("Error executing command: Ping " + strParams);
        EAScriptHost.LogError(e.message);

        // Any exceptions will result in the Remote Action failing
        RAContext.SetExecutionResult(RAContext.ExecutionError, "Error executing command: Ping " + strParams + ":" + e.message, -1);
    }

}

function ExecuteCommandWithResult(strCommand, strArgs) {

    // Execute the ping command via the Windows shell
    var WshShell = new ActiveXObject("WScript.Shell");
    var oExec = WshShell.Exec(strCommand + " " + strArgs);
    if (oExec == null) {
        EAScriptHost.LogError("ExecuteCommand: Could not execute command.");
        return -1;
    }

    // Read from the shell until execution is complete
    var allOut = "";
    var allError = "";
    var tryCount = 0;
    var tryReadCount = 0;
    while (tryCount < 60000) {

        var bRead = false;
        if (!oExec.StdOut.AtEndOfStream) {
            allOut += oExec.StdOut.ReadAll();
            bRead = true;
        }
        if (!oExec.StdErr.AtEndOfStream) {
            allError += oExec.StdErr.ReadAll();
            bRead = true;
        }

        if (!bRead) {
            if (tryReadCount++ > 10 && oExec.Status == 1)
                break;
            MMScriptHost.Sleep(100);
            tryCount += 100;
        }
        else {
            tryReadCount = 1;
            MMScriptHost.Sleep(1);
            tryCount++;
            EAScriptHost.LogDebug("ExecuteCommand: Reading data count=" + tryReadCount + ", Try Count=" + tryCount + ".");
        }

    }

    // If the execution timed out, log a message...
    if (tryCount >= 60000)
        EAScriptHost.LogError("ExecuteCommand: No response from process after 1min - aborting... Current Execution status=" +
oExec.Status);
```

```
// Log the result to the script's log file
if (allOut.length > 0)
    EAScriptHost.LogInfo(allOut);
if (allError.length > 0)
    EAScriptHost.LogError(allError);

// Set the result of the execution
if (oExec.ExitCode != 0) {
    EAScriptHost.LogError("ExecuteCommand: Application exited with code: " + oExec.ExitCode);
    RAContext.SetExecutionResult(RAContext.ExecutionError, allOut + allError, oExec.ExitCode);
}
else {
    EAScriptHost.LogInfo("ExecuteCommand: Application exited with code: " + oExec.ExitCode);
    RAContext.SetExecutionResult(RAContext.ExecutionOK, allOut, 0);
}

return oExec.ExitCode;
}
```

© Derdack GmbH | Derdack Corp., www.derdack.com

6   SAMPLE SCRIPTS

The following sample scripts are included in the default installation and are deactivated by default. All configuration settings to run the script are generally at the top of the script and need to be edited before being used e.g. if there is a specific user account that needs to be created or a path that needs to be set.

- **CreateMessages.js**: Enables the creation of all the types of messages available
- **HandleMessageEvents.js**: Enables the receipt of all the types of messages available and includes a function for reading message attachments. Each message is replied to if possible.
- **MonitorTicketStatus.js**: Demonstrates the ticket events **OnMemberNextMedia()**, **OnEscalationNextMember()** and **OnTicketStatus()**. Once a predefined escalation level has been reached, the specified user is notified.
- **ManageTicket.js**: Demonstrates basic ticket creation as well as most of the available functions for managing tickets
- **OpenNotepad.js**: Opens notepad when a SMS with the text "Notepad" arrives

7 OBJECT AND FUNCTION REFERENCE

7.1 Object Model Overview

The Scripting Host contains the following objects:
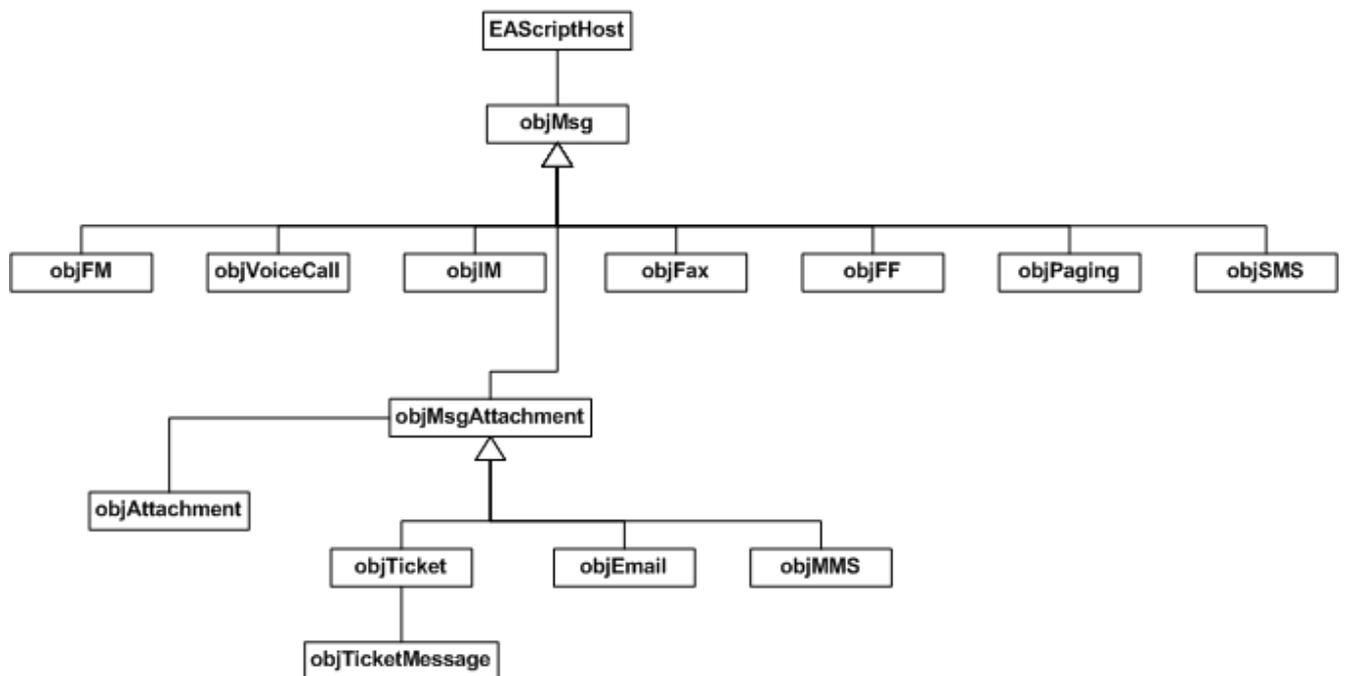
EAScriptHost is used for the creation of the base object types, from which other objects can be retrieved. The most important base object type being **objMsg**. **objMsg** essentially encapsulates an xml message that is sent or received from the Enterprise Alert® kernel.

The following diagram goes into further detail and lists all of the functions available through the object model:

objMsg
SetProperty() : Boolean
GetProperty() : String
HasProperty() : Boolean
GetMessageType() : String
GetXML() : String
CreateAnswer() : objMsg
GetMediaType() : String
Send() : Boolean

EAScriptHost
CreateActiveX() : Object
CreateSMS() : objSMS
CreateMMS() : objMMS
CreateEmail() : objEmail
CreateIM() : objIM
CreateVoiceCall() : objVoiceCall
CreatePaging() : objPaging
CreateFax() : objFax
CreateFM() : objFM
CreateFF() : objFF
TicketCreate() : objTicket
TicketDelete() : Boolean
TicketGetByUniqueId() : objTicket
TicketGetByExternalId() : objTicket
TicketGetByLastInternalId() : objTicket
TicketSetStatusToAcknowledged() : Boolean
TicketSetStatusToResolved() : Boolean
IsUserMemberOfAccount() : Boolean
GetProfileNameByAddress() : String
LogDebug()
LogInfo()
LogWarning()
LogError()

objSMS

objFM

objFax

objIM

objPaging

objVoiceCall

objFF

objMsgAttachment
ClearAttachments() : Boolean
GetNumberOfAttachments() : Integer
GetAttachment() : objAttachment
RemoveAttachment() : Boolean
AddAttachmentFiles() : Integer
AddAttachmentFilesByFolder() : Integer
AddAttachment() : Integer
SaveAttachments() : String
SaveAttachment() : String
SaveAttachmentsToMimeFile() : Boolean
GenerateSmil() : Boolean
SetSmilAutocreation()

objAttachment
GetName() : String
GetType() : String
GetFileSize() : Integer
GetMimeHeader() : String
GetMimeBody() : String

objMMS

objEmail

objTicket
GetUniqueTicketId() : Integer
GetInternalTicketId() : String
GetExternalTicketId() : String
GetTicketStatus() : Integer
GetCreatedOn() : String
GetClosedOn() : String
GetUserAcknowledgedBy() : String
GetUserResolvedBy() : String
GetTicketMessagesCount() : Integer
GetTicketMessageAt() : objTicketMessage

objTicketMessage
CreateNewMessage() : objMsg
GetId() : Integer
GetTicketMessageStatus() : Integer
GetUserName() : String
GetMediaType() : String

7.2 Object Member Details

7.2.1 Message Options

The following basic message options are supported and may only be applicable to certain message types:

- 0: Default message option. If you are not sure which option to use, then use this.
- 1: Deferred. Delivery of the message is deferred until a later stage. Typically applicable to SMS.
- 2: Direct display. The SMS is displayed directly on the mobile's display.
- 8: Binary. The content contains 8 bit binary hexadecimal characters.
- 16: User data header. The SMS contains user data header information.
- 64: Unicode. The message text contains Unicode characters
- 128: EMS. The message is an EMS e.g. a ringtone or formatted text.

7.2.2 Notification Types

For controlling the destination and notification type in a new created Ticket you can set the property notificationType to one of the following values.

- 0: Automatic. The alert destination will be determined automatically. If it is resolved as a Team, the alert will be processed as Team Escalation by default.
- 1: User alert. The alert destination is a User
- 2: Team Broadcast. The alert destination is a Team and the alert will be processed as Team Broadcast
- 3: Team Broadcast to a schedule. The alert destination is a scheduled Team and the alert will be processed as Team Broadcast
- 4: Team Escalation. The alert destination is a Team and the alert will be processed as Team Escalation
- 5: Team Escalation to a schedule. The alert destination is a scheduled Team and the alert will be processed as Team Escalation
- 6: Subscription Feed. The alert destination is a Subscription Feed

7.2.3 Ticket Status and Ticket Message Status

A ticket may have one of the following statuses:

- **None**: the ticket has not been initialized yet
- **Open**: the ticket has been initialized. Message(s) have been sent out to the associated recipients or are in the process of being sent out, but have not been acknowledged yet.
- **Acknowledged**: the ticket has been acknowledged by one of the recipient(s). The ticket status was previously "open".
- **Declined**: All of the recipient(s) to whom the ticket has been addressed have declined the ticket.
- **No Reply**: All of the recipient(s) to whom the ticket has been addressed have not responded to the ticket
- **Resolved**: the ticket has been resolved by one of the recipient(s). The ticket will go directly from a state of open to resolved if the priority of the ticket was low. If the priority of the ticket was major, the ticket becomes resolved, when the user responds to the initial ticket, otherwise if the priority is critical, the user needs to acknowledge the ticket before it can be resolved.
- **Failed**: None of the recipients could be sent a message or one or more of the message routes have failed.

- **Error**: An error has occurred during ticket processing.

Please refer to objTicket for a list of the ticket status properties and for further information as to how ticket statuses are used within scripting.

Ticket messages are encapsulated by **objTicketMessage**. Each ticket message is addressed to a specific user account. If the user has more than one media type specified and the message is addressed to FF (Find Me Follow Me), then when the user responds to a message, the response will only apply to the latest message sent out. Each ticket message can have the following statuses:

- **None**: no message has been sent to the recipient yet
- **Acknowledged**: the user has acknowledged the ticket message
- **Declined**: the user has declined the ticket message
- **No Reply**: the user has not responded to the message
- **Error**: there was an system error sending the message to the user
- **Failed**: the message could not be delivered to the user, possibly due to a message routing failure
- **Buffered**: the message has been sent, but is sitting at an intermediary system before being delivered to the user, or that the message may have been delivered, but the system is unable to determine whether the message has been delivered, and only that it has been successfully submitted to the intermediary system.
- **Delivered**: the message has been successfully delivered to the recipient
- **Not Delivered**: the message has not successfully been delivered to the recipient
- **Finished**: the user has resolved the ticket through the associated message
- **Answered**: the user has responded to the ticket message, but neither an acknowledgement nor a declination or resolution could be determined from the response.
- **Unavailable**: the destination's status is unavailable
- **Skipped**: the ticket message has been skipped and has not been sent to the recipient

Please refer to objTicketMessage for more information.

7.2.4 Media Types

The media type for an **objMsg** or **objTicketMessage** can be one of the following values:

- SMS
- MMS
- E-mail
- IM
- Voice-Call
- Paging
- Fax
- FF
- FM

7.2.5 EAScriptHost

object CreateActiveX(string strProgID)

Creates an activeX object by the given ProgID

objSMS CreateSMS()

Creates a new objSMS object

objMMS CreateMMS()

Creates a new objMMS object

objEmail CreateEmail()

Creates a new objEmail object

objIM CreateIM()

Creates a new objIM object

objVoiceCall CreateVoiceCall()

Creates a new objVoiceCall object

objPaging CreatePaging()

Creates a new objPaging object

objFax CreateFax ()

Creates a new objFax object

objFM CreateFM()

Creates a new objFM object

objFF CreateFF()

Creates a new objFF object

void LogDebug(string strText)

Writes strText to the log file (level Debug)

void LogInfo (string strText)

Writes strText to the log file (level Info)

void LogWarning (string strText)

Writes strText to the log file (level Warning)

void LogError (string strText)

Writes strText to the log file (level Error)

objTicket TicketCreate (strTo, strServiceTo, nPriority, strMessageText)

Creates a new ticket message **objTicket** with the destination account **strTo**, the destination service **strServiceTo**, the message priority **nPriority** and the message text **strMessageText**

boolean TicketDelete (int nUniqueTicketId)

The function deletes a ticket from Enterprise Alert® with the given unique ticket id. Returns true if the ticket was deleted successfully, otherwise false.

objTicket TicketGetByUniqueId(int nUniqueTicketId)

The function retrieves a ticket **objTicket** from the Enterprise Alert® database with the specified unique ticket id **nUniqueTicketId**. If the ticket cannot be found, the function returns null.

objTicket TicketGetByExternalId(string strExternalTicketId)

The function retrieves a ticket **objTicket** from the Enterprise Alert database with the specified external ticket id **strExternalTicketId**. If the ticket cannot be found, the function returns null.

objTicket TicketGetByLastInternalId(string strInternalTicketId)

The function retrieves the last ticket **objTicket** from the Enterprise Alert® database with the specified internal ticket id **strInternalTicketId**. There may be multiple tickets in the Enterprise Alert® database with the same internal ticket id, therefore the most recent ticket is retrieved. If the ticket cannot be found, the function returns null.

boolean TicketSetStatusToAcknowledged(int nUniqueTicketId, string strAcknowledgedBy, string strMediaType)

Sets the status of the ticket to acknowledge. If the ticket is successfully acknowledged, the function returns true, otherwise false. Only if the user is contained in the list of users to whom the ticket messages have been addressed, can the ticket be acknowledged. In addition, the ticket needs to have an open status i.e. it may not have already been acknowledged or have a closed status.

boolean TicketSetStatusToResolved(int nUniqueTicketId, string strResolvedBy, string strMediaType)

Sets the status of the ticket to resolved. If the ticket is successfully acknowledged, the function returns true, otherwise false. Only if the user is contained in the list of users to whom the ticket messages have been addressed, can the ticket be acknowledged. In addition, the ticket needs to have an open status i.e. it may not have already been acknowledged or have a closed status.

boolean IsUserMemberOfAccount(strUser, strAccount)

Returns true if the user account **strUser** belongs to the specified account **strAccount**, otherwise it returns false. The specified account refers to a group, an escalation chain, a group schedule or an escalation schedule.

string GetProfileNameByAddress(strAddress)

The message object **objMsg** as received from Enterprise Alert® through the events available through the Scripting Host will typically contain the address for the user account only and not the name of the user account e.g. if an SMS was sent to user Administrator, the **objMsg** object's **to** property will be set to +49123456789.

7.2.6 objTicket

Ticket status properties

objTicket has the following properties that return a value for the corresponding ticket status and can be compared to the value returned from **GetTicketStatus()**:**TicketStatusNone, TicketStatusOpen, TicketStatusResolved, TicketStatusFailed, TicketStatusError, TicketStatusAcknowledged, TicketStatusDeclined** and **TicketStatusNoReply.**

int GetTicketStatus ()

The function returns the ticket status for the current ticket object. Please refer to the ticket status properties for determining what the current ticket status is e.g.

```
if (objTicket.GetTicketStatus() == objTicket.TicketStatusDeclined)
{
    // Do something, because the ticket has been declined
}
```

int GetUniqueTicketId ()

The function returns the unique ticket id of the ticket. This id will only be populated for tickets received from Enterprise Alert® and not new tickets that have just been created.

string GetInternalTicketId ()

The function returns the internal ticket id of the ticket. This id will only be populated for tickets received from Enterprise Alert® and not new tickets that have just been created.

string GetExternalTicketId ()

The function returns the external ticket id of the ticket. The external ticket id may be something like a 3rd party application or custom id that has previously been set as a property on the message object. This value can also be retrieved as the message property externalTicketId.

string GetCreatedOn ()

Gets the date and time that the ticket was created on, otherwise returns a blank string

string GetClosedOn ()

Gets the date and time that the ticket was resolved on, otherwise returns a blank string

string GetUserAcknowledgedBy ()

The function returns the user that acknowledged the ticket, otherwise returns a blank string. The ticket status will therefore have already been set to **acknowledged**.

string GetUserResolvedBy ()

The function returns the user that acknowledged the ticket, otherwise returns a blank string. The ticket status will therefore have already been set to **acknowledged**.

int GetTicketMessagesCount ()

Gets the number of ticket messages associated with the ticket object. If the ticket messages are updated or the number of ticket messages change while the ticket object (**objTicket**) is being processed by the script, then the ticket object will need to be refreshed manually within the script.

objTicketMessage GetTicketMessageAt (int iTicketMessageIndex)

Returns a ticket message object with the given ticket message index. The ticket message index starts at 0 and is always less than the ticket message count returned by **GetTicketMessagesCount()**.

7.2.7 objTicketMessage

Ticket Message Status Properties

objTicketMessage has the following properties that return a value for the corresponding ticket message status and can be compared to the value returned from **GetTicketMessageStatus():TicketMessageStatusNone, TicketMessageStatusAcknowledged,**

© Derdack GmbH | Derdack Corp., www.derdack.com

TicketMessageStatusDeclined, TicketMessageStatusNoReply, TicketMessageStatusError, TicketMessageStatusFailed, TicketMessageStatusBuffered, TicketMessageStatusDelivered, TicketMessageStatusNotDelivered, TicketMessageStatusFinished, TicketMessageStatusAnswered, TicketMessageStatusUnavailable, TicketMessageStatusSkipped.

int GetTicketMessageStatus ()

Gets the ticket message status for the current ticket message object. Please refer to the ticket message status properties for determining what the current ticket message status is e.g.

```
if (objTicketMessage.GetTicketMessageStatus() == objTicketMessage.TicketMessageStatusFinished)
{
    // Do something, because the ticket message has been resolved for this ticket message
}
```

objMsg CreateNewMessage ()

The function returns an **objMsg** object. The type of message object (e.g. **objSMS**, **objMMS** etc.) will depend on the type of ticket message e.g. SMS, MMS etc. that was or is still to be sent to the recipient.

int GetId()

Returns the id of the ticket message in the Enterprise Alert® database

string GetUserName ()

Returns a string containing the user name of the recipient

string GetMediaType ()

The function returns a media type string for the current ticket message. Please refer to Media Types for more information.

7.2.8 objMsg

objMsg CreateAnswer()

Creates an answer **objMsg**

string GetProperty(string strPropName)

Returns the value of property **strPropName**

boolean SetProperty(string strPropName, string strValue)

The function sets a property value. Please refer to Get and set properties of message objects for a full list of all the properties that can be set.

boolean HasProperty(string strPropName)

Returns true if the property is available on the message object, otherwise false. A typical scenario where this might be useful is when the message type is unknown and a property needs to be accessed.

boolean Send()

The function sends the message to Enterprise Alert®. Returns true if successful, false if not.

string GetMessageType()

The function returns a string containing the message type. Possible values could be **userSMS, userMMS, userEmail, userIM, userPaging, userFax, userVoiceCall, userFF** and **userFM**. This function is **obsolete**. Please use string GetMediaType () instead.

string GetXML()

Returns an xml string for the message that is to be sent to or received from Enterprise Alert®

string GetMediaType ()

The function returns a media type string for the current ticket message. Please refer to Media Types for more information.

7.2.9 objMsgAttachment

Please refer to Handling attachments for more information.

bool ClearAttachments()

Clears all attachments

int GetNumberOfAttachments()

Returns the number of attachments

objAttachment GetAttachment(int nIndex)

Returns an **objAttachment** with index **nIndex**

bool RemoveAttachment(int nIndex)

Removes the attachment with index **nIndex**

int AddAttachmentFiles(string strPaths)

Adds an attachment by path

int AddAttachment(objAttachment objNewAtt)

Adds an attachment object

string SaveAttachments(string strDir)

Saves all attachments to a local directory

string SaveAttachment(string strDir, int nIndex)

Saves the attachment with the given index **nIndex** to the destination directory **strDir**

boolean SaveAttachmentsToMimeFile(string strDir, string strFileName)

Saves all attachments to a mime file with the name **strFileName** in directory **strDir**

boolean GenerateSmil()

Generates and adds a SMIL attachment. This is only applicable to MMS.

void SetSmilAutocreation(boolean bActivate)

This function enables/disables generation of SMIL part on sending. This is only applicable to MMS.

7.2.10 objAttachment

Please refer to <u>Handling attachments</u> for more information.

String GetName()

Returns the name of the attachment

String GetType()

Returns the MIME type of the attachment

int GetFileSize()

Returns the file size of the attachment

string GetMimeHeader ()

Returns a string with the attachment's mime header

string GetMimeBody ()

Returns a string with the attachment's mime body

7.2.11 Message Events

void OnSMS(objSMS msgObject)

When a SMS message is received by Enterprise Alert® *and* is passed onto the destination script, the above function is called. Additionally, an **objSMS** object passed as a parameter.

void OnMMS(objMMS msgObject)

When a MMS message is received by Enterprise Alert® *and* is passed onto the destination script, the above function is called. Additionally, an **objMMS** object passed as a parameter.

void OnEmail(objEmail msgObject)

When an email message is received by Enterprise Alert® *and* is passed onto the destination script, the above function is called. Additionally, an **objEmail** object passed as a parameter.

void OnIM(objIM msgObject)

When an instant message is received by Enterprise Alert® *and* is passed onto the destination script, the above function is called. Additionally, an **objIM** object passed as a parameter.

void OnVoiceCall(objVoiceCall msgObject)

When a voice call message is received by Enterprise Alert® *and* is passed onto the destination script, the above function is called. Additionally, an **objVoiceCall** object passed as a parameter.

void OnNewEvent (objNewEvent msgObject)

If a message is received by Enterprise Alert® and does not fall into the above categories and has been passed onto the destination script, the above function is called. You can query the object type information

using such functions as **objMsg.GetMediaType()** and **objMsg.HasProperty()** etc.

7.2.12 Ticket Events

void OnTicketStatus(int nTicketStatus, int nUniqueTicketId, string strInternalTicketId, string strExternalTicketId)

Each time the ticket status changes this event is fired. To implement this event, simply implement a function with the same name and parameters.

void OnEscalationNextMember(string strMemberName, int nUniqueTicketId)

Each time a message associated with a ticket is submitted to a user account within an escalation chain, this event is fired. To implement this event, simply implement a function with the same name and parameters.

void OnMemberNextMedia(string strUserName, string strMediaType, int nUniqueTicketId)

Each time a message associated with a ticket is submitted to a user account within an escalation chain, this event is fired. Each user may have multiple media set up in their media chain, which means that if the user does not respond with one medium, then a message is sent to next medium in the media chain. To implement this event, simply implement a function with the same name and parameters.

7.2.13 RAContext

RAContext is used for setting the result of a Remote Action's execution.

void SetExecutionResult(int nStatus, string strStatusDescription, int nErrorCode)

nStatus needs be either set as **ExecutionOK** or **ExecutionError** depending on the result the execution. **strStatusDescription** specifies a text result for the execution, which can be viewed by the user in the Mobile App or in the Remote Action execution details. nErrorCode can be any error code that you define or can be for example the error code returned by an external application called by your function.

8 SUPPORT / CONTACT

8.1 Support

When purchasing Enterprise Alert® you automatically are granted a period of 30-days of free support via email, fax or telephone.

Please, note that such support cannot replace any need for a professional software deployment in enterprise IT environments. In any case we recommend to make use of our deployment team for a rapid and seamless deployment of Enterprise Alert®.

After your free support time has expired we offer the following support options:

> Support and maintenance agreement:
> For business critical applications that require 100% reliability and system availability DERDACK offers knowhow and expertise within an optional support and service agreements. In order to provide you with a highly professional and responsive service in case of occurring problems, you want a partner that reacts and solves the issue for you fast. We thus strongly recommend a support and service agreement with us which regulates e.g. the following services:

- 9x5 business hour or 24x7 around-the-clock premium support
  Extended liabilities
- Preferred onsite-services
- Rebates on consulting services, etc.

For more information on our support and maintenance agreements please contact our sales team (sales@derdack.com).

8.2 End User License Agreement (EULA)

For the SOFTWARE PRODUCTS described in the manual and marked © Derdack GmbH the following license agreement applies to you:

This is a legal agreement between you, the end user, and Derdack GmbH, Germany (called "Derdack" hereafter"). By installing this SOFTWARE PRODUCT onto your PC or server, you are agreeing to be bound by the terms of this agreement. If you do not agree to the terms of this agreement, promptly stop the installation process by clicking the "Cancel" button and return the product package or license file to the place you obtained it for refund.

1. Grant of license
Derdack grants you the right to use one copy of the enclosed SOFTWARE PRODUCT - the SOFTWARE - on a single computer. The SOFTWARE is in use on a computer when it is loaded into memory or installed into permanent memory of that computer. However, installation on a network server for the sole purpose of distribution to one or more computer(s) shall not constitute "use" for which a separate license is required. It is expressly understood by Derdack and the end user that other than stated here in this Agreement no further rights and/or ownership in the SOFTWARE PRODUCT are granted to the end user.

2. Copyright
This software is owned by Derdack or its suppliers and is protected by German and international copyright laws and treaties. Therefore you must treat the SOFTWARE like any other copyrighted material. Except that if the SOFTWARE is not copy protected you may either make one copy of the SOFTWARE PRODUCT solely for backup purpose or transfer the SOFTWARE to a single hard disk provided that you keep the original for

backup or archive purposes. You may not copy the product manuals or any written material accompanying the SOFTWARE PRODUCT.

3. Other restrictions

You may not rent or lease the SOFTWARE, but you may transfer your rights under this license agreement on a permanent basis if you transfer all copies of the SOFTWARE PRODUCT and all written material, and if the recipient agrees to the terms of this agreement. You may not reverse engineer, de-compile or disassemble the SOFTWARE. Any transfer must include the most recent update and all prior versions.

4. Limited warranty

Derdack warrants that the SOFTWARE will perform substantially in accordance with the accompanying printed or electronic product manual(s) for the shortest period permitted by applicable law but a minimum of ninety (90) days.

Warranty implies that Derdack shall, within a reasonable timeframe, either (at Derdack's option) repair, replace, support, remedy or take other corrective or error avoidance action as may be necessary to correct all reported Defects. The obligation of this Article is however subject to the following conditions:

- The Defect must constitute an operating defect compared to the Specification
- The End user must inform Derdack of the Defect in writing
- The part of the System concerned must not have been:
  - o modified by the End user without Derdack's written authorisation
  - o installed, used or maintained in a manner contrary to Derdack's reasonable instructions
  - o subjected to misuse (including for purposes other than for which they were designed) improper maintenance, unusual workload and accident
  - o damaged by deviation from applicable environmental specifications
  - o subject to vandalism, burglary or theft.

In the event of any claim of Defect by the End user under the above warranties, the End user shall inform Derdack of the nature of any alleged defect whereupon Derdack or its representative shall have the option of testing and inspecting the System by remote diagnosis or at the End user site (provided it complies with the End user's security rules). The End user shall ensure that Derdack is provided with all information regarding the alleged defect which Derdack may reasonably require and such reasonable assistance as Derdack may request when carrying out any investigation.

5. End user remedies

Derdack's entire liability and your exclusive remedy shall be, at Derdack's option, either

- a return of the price paid or
- repair or replacement of the SOFTWARE that not meets the limited warranty and which is returned with a copy of your receipt.

The limited warranty is void if failure of the SOFTWARE has resulted from accident, abuse or misapplication. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period but at least for 30 days.

6. No other warranties

To the maximum extent permitted by applicable law, Derdack disclaims all other warranties, either express of implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the SOFTWARE, the accompanying product manual(s) and written materials. The limited warranty contained herein gives you specific legal rights.

7. No liability for consequential damage

To the maximum extent permitted by applicable law, Derdack and its suppliers shall not be liable for any other damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use this SOFTWARE PRODUCT, even if Derdack has been advised of the possibility of such damages. In any case, Derdack's entire liability under any provisions of this agreement shall be limited to the amount actually paid by you for this SOFTWARE PRODUCT.

8. No liability for message delivery or receipt

Derdack cannot guarantee that (mobile) messages sent through the used mobile or any other equipment by the SOFTWARE PRODUCT reach their destination. Neither can Derdack guarantee that the SOFTWARE PRODUCT receives all messages through the used mobile equipment they have been sent to. Derdack is not liable for any consequential damage arising from the fact that messages sent by the SOFTWARE PRODUCT do not reach their target destination (mobile phones, pagers) or that messages sent from any mobile equipment to the SOFTWARE PRODUCT will be recognized and processed by the SOFTWARE PRODUCT.

9. Terms

This Agreement will continue in effect for the duration of usage of the SOFTWARE by the end user.

10. Governing Law

This Agreement will be governed and interpreted in accordance with the laws of the Federal Republic of Germany. Venue for any legal action shall lie with the courts of Berlin, Germany, and the End user and Derdack each expressly consent to venue and jurisdiction therein.

11. Severability

If any of the provisions of this Agreement are found or deemed by a court of competent jurisdiction to be invalid or unenforceable, they shall be severable from the remainder of the Agreement and shall not cause the invalidity or unenforceability of the Agreement

12. Other License Agreement

This License Agreement may be replaced by a separate software license agreement agreed upon between the end user and Derdack.

8.3 About

Derdack designs premium software for business-critical mobile messaging that empowers enterprises to gain vital competitive advantages. Derdack is the provider of Enterprise Alert® - a leading portfolio of SMS & MMS application platforms and enterprise-class software for automation of business-critical alerts.

Derdack's messaging platform products enable rapid and effortless design and introduction of revenue-generating mobile messaging services. Derdack's enterprise product provides efficient disaster preventation and accelerated response to business-critical situations.

Derdack is a privately held, cash-flow financed company and has been a leading expert in the mobility area since 1999. Derdack has several hundred installations worldwide including BMW UK, Caterpillar Belgium, Proximus Belgium, Roche Switzerland, Siemens Germany, Movil de Entel Bolivia, Orascom Tunisia, Telstra Australia, Microsoft Ireland, Telenor Norway, Steria UK, Daimler Chrysler Germany, RasGas Qatar, Etisalat Dubai, Oxy Petrol Qatar.

Derdack is partnering with Microsoft, Hewlett-Packard and IBM and has reseller network covering more than 30 countries worldwide.

8.4 Contact

If you have any questions regarding our messaging software or about the company, please, feel free to contact us:

Derdack GmbH
Wilhelmgalerie
Friedrich-Ebert-Strasse 8
14467 Potsdam
Germany

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

www.derdack.com