Enhydra Shark Tool Agents

Table of Contents

About tool agents (quotation from WfMC document)

The "Invoking Applications Interface" defines an interface mechanism between Workflow Management Systems and any other application, but it, however, differentiates itself from the other Coalition interface definitions. Invoking an application is not a workflow specific functionality, but a Workflow System would not make much sense without this functionality.

Therefore, this interface addresses workflow system vendors as well as any third party software vendor. Based on different communication technologies the so-called "Tool Agents" can handle the application control and information exchange. These Tool Agents represent at least one specific invocation technology. E.g. while one Tool Agent supports DDE commands, others can communicate based on protocols like OLE or CORBA or any other concept.

The technology to interact between a Tool Agent and a corresponding application depends on the underlying architecture and on application - specific interfaces, which have to be managed under control of the Tool Agent itself. The suggested interface defines the way a Tool Agent can be used by a workflow application, e.g. a worklist handler or the workflow engine. Finally, the purpose of Tool Agents can be compared with the purpose of standardized software components.

The set of application interface functions provides services to Tool-Agents, to invoke and control applications associated with specific work items.

The Invoked Application Interface defines an API set, which is highly recommended to be used by Workflow System components (engine and client applications) to control specialized application drivers called Tool Agents. These Tool Agents finally start up and stop applications, pass workflow and application relevant information to and from the application and control the application's run level status. Therefore, the Invoked Application Interface WAPIs are only directed against a Tool Agent. Nevertheless, additional workflow information could be requested by an application via the Tool Agent using standard WAPI functions. As the Invoked Application Interface should handle bi-directional requests (requests to and from applications), it depends on the interfaces and architecture of applications how to interact with an Tool Agent.

This interface will allow the request and update of application data and more run-time relevant functionalities.

The Workflow System itself has to know about the installed Tool Agents. The basic architecture of Tool Agents could be compared with a driver - interface, i.e. ODBC, etc.. Within this interface definition, no further communication mechanism between the Tool Agents and the Workflow System is necessary.

Shark Implementation of Tool Agent Interface

Shark defines Tool Agent interface to be its internal interface - clients know nothing about it.

We defined our own Java interface for tool agents, and this interface is based on WfMC specification. It is defined in SharkAPI module, in the package *org.enhydra.shark.internal.toolagent*. There is a pretty good documentation of API for tool agents, and it can be found in documentation's tool agent api section [../api/SharkAPI/org/enhydra/shark/api/internal/toolagent/package-summary.html]

How does Shark Use Tool Agents

Shark knows only about tool agent interface. It doesn't know anything about any particular tool agent implementation. When automatic* activity of "Tool" type is performed, shark searches for the appropriate tool agent:

- if mapping information for the application definition which is referenced from this activity's tool exists (Admin has previously mapped the application definitions to tool agents), shark finds this mapping, and gets the information which tool agent to call, and what are the mapping parameters to be passed to tool agent. Shark then tries to connect tool agent, and gets a handle to it. Then it calls invokeApplication() method of tool agent, and passes the relevant parameters (some of them contained in the mapping information, more precisely the application name and application mode parameter). After that, it calls its method requestAppStatus(), to check the tool agent's status and to retrieve the results, and set appropriate activity variables.

- If mapping information does not exist, shark calls **"Default tool agent"**, whose class name is specified in shark's configuration (i.e in file Shark.conf if shark is configured through the file), and does the same as previously mentioned.

When calling tool agent's invokeApplication() method, for the first AppParameter in the AppParameter array, shark is always passing a string representing ExtendedAttributes section of corresponding XPDL application, chopped out from the XPDL definition, i.e.:

```
<ExtendedAttributes>
   <ExtendedAttribute Name="ToolAgentClass"
 Value="org.enhydra.shark.toolagent.JavaScriptToolAgent"/>
   <ExtendedAttribute Name="Script" Value="c=a-b;"/>
</ExtendedAttributes>
```

As previously mentioned, if shark can't found mapping information, it executes **Default tool agent**. The default tool agent is responsible to execute proper tool agent if it finds in ExtendedAttributes information which tool agent to execute. Default tool agent gets this information from XPDL application extended attribute whose name has to be **ToolAgentClass** and value has to be the full class name of wanted tool agent. Other extended attributes are supposed to be read by tool agent specified to be executed, and are "Tool agent specific".

* NOTE: Shark automatically starts "Tool" activities under following conditions:

1. If "Tool" activity's Start mode is AUTOMATIC (if Start mode is not defined, the automatic mode is assumed)

2. If "Tool" activity has a performer different then "SYSTEM" type participant, and its Start mode is MANUAL

In the second case, the activity will first be assigned to participant, and after he completes activity, the tools specified will be executed automatically by the shark, through tool agent calls.

Shark Tool Agent Examples

There are several useful general purpose tool agents comming with Shark. They can serve as an example how to develop your own, probably more complex tool agents. The source code for our tool agents can be found in modules/SharkToolAgent.

- JavaClassToolAgent - This tool agent executes Java classes, by calling its static method called "execute". When calling this tool agent's invokeApplication() method, the application name parameter should be the full name of the class that should be executed by this tool agent. So far, we defined a few classes that execute simple arithmetic operation, generation of random number, and one that performs waiting. There are also two classes contributed to by Paloma Trigueros Cabezon, and they can be used to use this tool agent to send mails.

  This tool agent is able to "understand" the extended attribute with the following name:

  - AppName - value of this attribute should represent the class name to be executed

  The tool agent will pass all the parameters it gets (instance variables described by the formal parameters defined in the XPDL application definition) to the Java class it is executing.

- RuntimeApplicationToolAgent - Executes system executables like notepad on Windows or Vi editor on Unix system. The application **MUST** be in the system path of machine where shark is running.

  If you use application mode 0 (zero), the tool agent will wait until the executable application is completed, and if you choose application status other then 0 the tool agent will finish its work as soon as the executable application is started (this usually happens immediately), and shark will proceed to the next activity, even if the executable application is still running (this is asynchronous execution of applications).

  This tool agent is able to "understand" extended attributes with following names:

  - AppName - value of this attribute should represent the executable application name to be executed by tool agent

  - AppMode - value of this attribute should represent the mode of execution, if set to 0 (zero), tool agent will wait until the executable application is finished.

  The tool agent accepts parameters (AppParameter class instances), but does not modify any. The parameters for which the corresponding application definition formal parameters are "IN" type and whose data type is string are added as suffixes to the application name, and resulting application (command) that is executed could be something like "notepad c:\Shark\readme" ('c:\Shark\readme' is the parameter provided).

- JavaScriptToolAgent - Executes java script. If you set application mode to 0 (zero), tool agent will search for a java script file given as applicationName parameter (this file has to be in the class path), and if it finds it, it will try to execute it. Otherwise, it will consider applicationName parameter to be the script itself, and will try to execute it. So far, to show an example, we defined few java script files that execute simple arithmetic operations, generation of random number, and one that performs waiting.

  This tool agent is able to "understand" the extended attributes with the following names:

- AppName - if present, the value of this attribute should represent the name of script file to execute (this file has to be in class path)

- Script - the value of this parameter represents the script to be executed. I.e. this extended attribute in XPDL can be defined as follows:

```
<ExtendedAttribute Name="Script" Value="c=a-b;"/>
```

  (a, b and c in above text are Formal parameter Ids from XPDL Application definition)

The tool agent will provide all the parameters it gets (instance variables described by the formal parameters defined in the XPDL application definition) to the Java script interpreter so it can evaluate script.

- BshToolAgent - Executes script written in Java language syntax. If you set application mode to 0 (zero), tool agent will search for a script file given as applicationName parameter (this file has to be in the class path), and if it finds it, it will try to execute it. Otherwise, it will consider applicationName parameter to be the script itself, and will try to execute it.

  This tool agent is able to "understand" the extended attributes with the following names:

  - AppName - if present, value of this attribute should represent the name of script file to execute (this file has to be in class path)

  - Script - the value of this attribute represents the script to be executed. I.e. this extended attribute in XPDL can be defined as follows:

```
<ExtendedAttribute Name="Script" Value="c=new
 java.lang.Long(a.longValue()-b.longValue());"/>
```

    (a, b and c in above text are Formal parameter Ids from XPDL Application definition)

  The tool agent will provide all the parameters it gets (instance variables described by the formal parameters defined in the XPDL application definition) to the Java expression interpreter so it can evaluate script.

- XSLTToolAgent - applies xsl transformation to the provided String, byte[] or XML variable, and produces String, byte[] or XML variable as a result.

  XSLTToolAgent is able to understand only variables (formal parameters) with a certain Ids and these variables have the special meaning and the order of the variables (the order of formal parameters defined in XPDL) is not important. The following is description of all possible variables/formal parameters this tool agent can interpret:

  - source - value of this attribute represents the source of transformation and it can be defined as String, byte[] or Schema formal parameter in XPDLs application definition.

  - result - value of this attribute represents the result of transformation and it can be defined as String, byte[] or Schema formal parameter in XPDLs application definition.

  - transformer_name - value of this attribute represents the name of XSL transformation which must be present in the classpath. It must be defined as String formal parameter in XPDLs application definition.

  - transformer_path - value of this attribute represents the location of XSL transformation on the file system (it is used only if there is no transformer_name formal parameter). It must be defined as String formal parameter in XPDLs application definition.

- transformer_node - This parameter must be defined as Schema formal parameter in XPDLs application definition. It is an XML representing the XSL transformation(it is used only if there is no transformer_name and transformer_path formal parameters defined).

- transformer_script - This parameter must be defined as String formal parameter in XPDLs application definition. It is a string representing the XSL transformation(it is used only if there is no transformer_name, transformer_path and transformer_node formal parameters defined).

The tool agent also understands a special extended attribute of XPDLs application definition with name "Script". The value of this attribute represents the XSL transformation to be executed and will be used only if there is no none of the following formal parameters defined for the application definition: transformer_name, transformer_path, transformer_node, transformer_script.

If there are other then above mentioned formal parameters defined in XPDLs application definition, they will be passed as a parameters to the XSL transformation.

- SOAPToolAgent - Executes WEB service operation.

  When you map XPDL application to this tool agent, you should set application name to the location of the WSDL file that defines WEB service to be called.

  This tool agent is able to "understand" the extended attribute with the following name:

  - AppName - value of this attribute should represent the location of WSDL file where WEB service is defined.

  This tool agent requires that the first parameter defined in XPDL Application's formal parameters represent the name of WEB service operation to be called. The tool agent will include all other parameters it gets (instance variables described by the formal parameters defined in the XPDL application definition) in the WEB Service call.

- MailToolAgent - sends and receives mail messages.

  There is a MailMessageHandler interface defined that is used to actually handle mails. We provided default implementation of this interface, but one can create its own implementation. This interface is specifically defined for this tool agent, and is not a part of Shark's APIs.

  Beside default implementation of MailMessageHandler represented by class DefaultMailMessageHandler, another implementation is available, SMIMEMailMessageHandler. Actually, it is an extension of the default implementation. The SMIMEMailMessageHandler enables sending encrypted messages, signed messages, or encrypted and signed messages, according to SMIME specification. More about this handler will be explained later.

  When performing mappings, you should set application name to be the full class name of the implementation class of MailMessageHandler interface.

  To be able to work with our DefaultMailMessageHandler, you must define some properties, and here is a section from shark's configuration file "Shark.conf" that defines these properties:

```
#
# the properties for our default implementation of MailMessageHandler interface
# required by MailToolAgent
#

# the parameters for retrieving mails, possible values for protocol are "pop3" and "imap"
DefaultMailMessageHandler.IncomingMailServer=someserver.co.yu
DefaultMailMessageHandler.IncomingMailProtocol=pop3
DefaultMailMessageHandler.StoreFolderName=INBOX
```

```
DefaultMailMessageHandler.IMAPPortNo=143
DefaultMailMessageHandler.POP3PortNo=110

# the parameters for sending mails
DefaultMailMessageHandler.SMTPMailServer=someserver.co.yu
DefaultMailMessageHandler.SMTPPortNo=25
DefaultMailMessageHandler.SourceAddress=shark@objectweb.org

# credentials
DefaultMailMessageHandler.Login=shark
DefaultMailMessageHandler.Password=sharkspwd

# authentication
DefaultMailMessageHandler.useAuthentication=false
```

This tool agent is able to "understand" the extended attributes with the following names:

- AppName - value of this attribute should represent the full class name of MailMessageHandler interface implementation that will handle mails. To use our default implementation, specify the value "org.enhydra.shark.toolagent.DefaultMailMessageHandler".

- AppMode - value of this attribute should represent the mode of execution, if set to 0 (zero), tool agent will send mails, and if set to 1 it will receive mails.

The tool agent will provide all the parameters it gets (as described by the formal parameters defined in the XPDL application definition) to the mail message handler.

Default mail message handler is able to understand only STRING variables (formal parameters) with a certain Ids and these variables have the special meaning and the order of the variables (the order of formal parameters defined in XPDL) is not important. The following is description of all possible variables/formal parameters this mail handler can interpret:

- from_addresses - value of this attribute should be comma separated string representing address(es) of the mail sender(s)

- from_names - value of this attribute should be comma separated string representing name(s) of the mail sender(s).

- to_addresses - value of this attribute should be comma separated string representing to address(es) of the mail recepient(s).

- to_names - value of this attribute should be comma separated string representing to name(s) of the mail recepient(s).

- cc_addresses - value of this attribute should be comma separated string representing cc address(es) of the mail recepient(s).

- cc_names - value of this attribute should be comma separated string representing cc name(s) of the mail recepient(s).

- bcc_addresses - value of this attribute should be comma separated string representing bcc address(es) of the mail recepient(s).

- bcc_names - value of this attribute should be comma separated string representing bcc name(s) of the mail recepient(s).

- subject - value of this attribute defines the mail subject.

- content - value of this attribute defines the mail content.

- charset - value of this attribute defines the charset to be used for from/to/cc/bcc/subject and for the text of non-multipart mails that do not have mime_type attribute defined.

- mime_type - value of this attribute defines the mime type of the mail content.

- file_attachments - value of this attribute should be comma separated string representing absolute path(s) to the file(s) that will be send as the mail attachment(s).

- file_attachments_names - value of this attribute should be comma separated string representing names used for attached files. If such attribute does not exist, the name of the corresponding file will be used to specify attachment name.

- url_attachments - value of this attribute should be comma separated string representing URL(s) that will be send as the mail attachment(s).

- url_attachments_names - value of this attribute should be comma separated string representing names used for attached urls. If such attribute does not exist, the name of the corresponding url will be used to specify attachment name.

- var_attachments - value of this attribute should be comma separated string representing byte[] variable(s) Id(s) from the process context that will be send as the mail attachment(s). The byte[] represents the file serialized into shark's database.

- var_attachments_names - value of this attribute should be comma separated string representing names used for attached content. If such attribute does not exist, the name of the corresponding variable will be used to specify attachment name.

- var_attachments_mime_types - value of this attribute should be comma separated string representing mime type(s) for the byte[] variable(s) specified by the previously described attribute.

In order to send an email, the minimal requirement is to have at least one of the attributes determining **to**, **cc** or **bcc** recepeints defined (typically it will be only **to_addresses** attribute defined)

To be able to work with our SMIMEMailMessageHandler, beside properties defined for default implementation (described above), some additional properties should be defined. Note that those properties are default properties which will be used instead or missing variables/formal parameters, or in correlation with existing variables/formal parameters. Here is a section from Shark's configuration file "Shark.conf" that define properties considered SMIME:

```
#
# The default parameters used for SMIME implementation of MailMessageHandler
# interface required by MailToolAgent
#

# default enveloping parameters
SMIMEMailMessageHandler.Env.Default.Path=
SMIMEMailMessageHandler.Env.Default.KeystoreName=
# Allowable values are: BKS, JKS, PKCS12, UBER
SMIMEMailMessageHandler.Env.Default.KeystoreType=JKS
SMIMEMailMessageHandler.Env.Default.KeystorePassword=
# Allowable values are: DES(key length 56), DES_EDE3_CBC(key length 128,192), RC2_CBC (key
 length 40, 64, 128)
SMIMEMailMessageHandler.Env.Default.Algorithm=RC2_CBC
SMIMEMailMessageHandler.Env.Default.KeyLength=40

# default signing parameters
SMIMEMailMessageHandler.Sig.Default.Path=
SMIMEMailMessageHandler.Sig.Default.KeystoreName=
# Allowable values are: BKS, JKS, PKCS12, UBER
```

```
SMIMEMailMessageHandler.Sig.Default.KeystoreType=JKS
SMIMEMailMessageHandler.Sig.Default.KeystorePassword=
# Allowable values are: MD2_WITH_RSA, MD5_WITH_RSA, SHA1_WITH_DSA, SHA1_WITH_RSA
SMIMEMailMessageHandler.Sig.Default.Algorithm=SHA1_WITH_RSA
SMIMEMailMessageHandler.Sig.Default.IncludeCert=True
SMIMEMailMessageHandler.Sig.Default.IncludeSignAttrib=True
SMIMEMailMessageHandler.Sig.Default.ExternalSignature=True
```

Parameters are devided in two groups: enveloping (encrypting) parameters and signing parameters. Theoretically, all of SMIME configuration parameters are optional and will be used only if properties are not defined via variables/formal parameters. Practically, to avoid repetition of values in variables/formal parameters, it is advisable to put some properties to 'default level' - it means configuration file.

- SMIMEMailMessageHandler.Env.Default.Path - the default directory path considered as root point for organisation of certificate (.cer) files or/and Java 'key store' files with certificates. This path is used as path prefix for variables/formal parameters that points to certificate (.cer) files or/and Java 'key store' files which are not defined by absolute path.

- SMIMEMailMessageHandler.Env.Default.KeystoreName - the name of the default Java 'key store' which will be used if corresponding variables/formal parameter for Java 'key store' file with certificates is missing. Note that this file should be placed in default directory defined by previous parameter.

- SMIMEMailMessageHandler.Env.Default.KeystoreType - the default type of Java 'key store' which will be used if corresponding variables/formal parameter is missing. The allowable values are: BKS, JKS, PKCS12, UBER.

- SMIMEMailMessageHandler.Env.Default.KeystorePassword - the default password that enables access to Java 'key store' which will be used if corresponding variables/formal parameter is missing.

- SMIMEMailMessageHandler.Env.Default.Algorithm - the default symetric algorithm for enveloping process which will be used if corresponding variables/formal parameter is missing.

- SMIMEMailMessageHandler.Env.Default.KeyLength - the default key length for symetric algorithm for enveloping preocess which will be used if corresponding variables/formal parameter is missing.

- SMIMEMailMessageHandler.Sig.Default.Path - the default directory path considered as root point for organisation of certificate with private key files (.pfh, .p12) or/and Java 'key store' files with certificates and private keys. This path is used as path prefix for variables/formal parameters that points to certificate with private key files (.pfh, .p12) or/and Java 'key store' files with certificates and private keys which are not defined by absolute path.

- SMIMEMailMessageHandler.Sig.Default.KeystoreName - the name of the default Java 'key store' that will be used if corresponding variable/formal parameter for Java 'key store' file with certificates and private keys is missing. Note that this file should be placed in default directory defined by previous parameter.

- SMIMEMailMessageHandler.Sig.Default.KeystoreType - the default type of Java 'key store' which will be used if corresponding variables/formal parameter is missing. The allowable values are: BKS, JKS, PKCS12, UBER.

- SMIMEMailMessageHandler.Sig.Default.KeystorePassword - the default password that enables access to Java 'key store' which will be used if corresponding variables/formal parameter is missing.

- SMIMEMailMessageHandler.Sig.Default.Algorithm - the default asymetric algorithm for signing process which will be used if corresponding variables/formal parameter is missing.

- SMIMEMailMessageHandler.Sig.Default.IncludeCert - the default decision to include signer's certificate chain into signed message or not include. This parameter will be used if corresponding variables/formal parameter is missing.

- SMIMEMailMessageHandler.Sig.Default.IncludeSignAttrib - the default decision to include signing attribute into signed message or not include. This parameter will be used if corresponding variables/formal parameter is missing.

- SMIMEMailMessageHandler.Sig.Default.ExternalSignature - the default decision what kind of signing will be: external or internal. This parameter will be used if corresponding variables/formal parameter is missing.

SMIME mail message handler is able to understand only the STRING variables (formal parameters) with a certain Ids and these variables have the special meaning and the order of the variables (the order of formal parameters defined in XPDL) is not important. The following is description of all possible variables/formal parameters this mail handler can interpret:

- SecurityType - value of this attribute represents choosen security type for email that will be send. The parameter is of String type and can take the following values: 1 - SignedSMIME, 2 - EnvelopedSMIME, 3 - SignedAndEnvelopedSMIME, 4 - EnvelopedAndSignedSMIME. Anything else means that there is no security issues and pure email will be sent (like with DefaultMailMessageHandler)

- Env_TO_Cert - value of this attribute should be comma separated string representing array of certificates (.cer files) which correspond to recipients marked as 'TO' recipients (recipients given by 'to_addresses' attribute). Certificates are used for enveloping (encrypting) of message. The certificates can be represented by their absolute paths, by their relative paths, or by their names only. In last two cases the default path from configuration file (parameter 'SMIMEMailMessageHandler.Env.Default.Path') will be added as prefix to certicicates. The combination of all of this certificate definitions can be used as array items. Note that number of array items must be equal to number of 'TO' recipients given via 'to_addresses' attribute. If any certificates are missing, they should be defined as empty items in array of certificates (items with one or more space characters). Missing certificates then must be found from 'Key Store' definition. There are two parallel ways to define certificates (via path to .cer files and from defined 'Key Stores'). All certificates can be defined on either one of those ways, or they can be defined combined both.The count of certificates, no matter how they are defined, must be equal to count of 'TO' recipients.

- Env_TO_Keystore - value of this attribute should be comma separated string representing array of 'Key Stores' files which correspond to recipients marked as 'TO' recipients (recipients given by 'to_addresses' attribute). Java 'Key Store' keep certificates which are used for enveloping (encrypting) of message. The 'Key Stores' can be represented by their absolute paths, by their relative paths, or by their names only. In last two cases the default path from configuration file (parameter 'SMIMEMailMessageHandler.Env.Default.Path') will be added as prefix to 'Key Store'. The combination of all of this definitions can be used as array items. Note that number of array items must be equal to number of 'TO' recipients given via 'to_addresses' attribute. If some 'Key Stores' are missing they should be defined as empty items in array of 'Key Stores' (items with one or more space characters). Missing 'Key Stores' then must be found from certificate definition (argument declared above). There are two parallel ways to define certificates (via path to .cer files and from defined 'Key Stores'). All certificates can be defined on either one of those ways, or they can be defined combined both.The count of certificates, no mether how they are defined, must be equal to count of 'TO' recipients.

- Env_TO_KeystoreType - value of this attribute should be comma separated string representing array of 'Key Store' types, which correspond to recipients marked as 'TO' recipients (recipients given by

'to_addresses' attribute). Allowable values are: BKS, JKS, PKCS12, UBER. Note that number of array items must be equal to number of 'TO' recipients given via 'to_addresses' attribute. If some items are missing, they should be defined as empty items in array (items with one or more space characters). Missing 'Key Store' types then must be found from default type definition placed in Shark configuration file (parameter SMIMEMailMessageHandler.Env.Default.KeystoreType).

- Env_TO_KeystorePassword - value of this attribute should be comma separated string representing array of 'Key Store' passwords, which correspond to recipients marked as 'TO' recipients (recipients given by 'to_addresses' attribute). Note that number of array items must be equal to number of 'TO' recipients given via 'to_addresses' attribute. If some items are missing, they should be defined as empty items in array (items with one or more space characters). Missing 'Key Store' passwords then must be found from default password definition placed in Shark configuration file (parameter SMIMEMailMessageHandler.Env.Default.KeystorePassword).

- Env_TO_KeystoreCertAlias - value of this attribute should be comma separated string representing array of 'Key Store' certificate aliases which correspond to recipients marked as 'TO' recipients (recipients given by 'to_addresses' attribute). Aliases are used to find desired certificate from 'KeyStore'. Note that number of array items must be equal to number of 'TO' recipients given via 'to_addresses' attribute.

- Env_CC_Cert - value of this attribute should be comma separated string representing array of certificates (.cer files) which correspond to recipients marked as 'CC' recipients (recipients given by 'cc_addresses' attribute). For more information read about Env_TO_Cert argument, whose functionality is quite similar.

- Env_CC_Keystore - value of this attribute should be comma separated string representing array of 'Key Stores' files which correspond to recipients marked as 'CC' recipients (recipients given by 'cc_addresses' attribute). For more information read about Env_TO_Keystore argument, whose functionality is quite similar.

- Env_CC_KeystoreType - value of this attribute should be comma separated string representing array of 'Key Store' types, which correspond to recipients marked as 'CC' recipients (recipients given by 'cc_addresses' attribute). For more information read about Env_TO_KeystoreType argument, whose functionality is quite similar.

- Env_CC_KeystorePassword - value of this attribute should be comma separated string representing array of 'Key Store' passwords, which correspond to recipients marked as 'CC' recipients (recipients given by 'cc_addresses' attribute). For more information read about Env_TO_KeystorePassword argument, whose functionality is quite similar.

- Env_CC_KeystoreCertAlias - value of this attribute should be comma separated string representing array of 'Key Store' certificate aliases which correspond to recipients marked as 'CC' recipients (recipients given by 'cc_addresses' attribute). For more information read about Env_TO_KeystoreCertAlias argument, whose functionality is quite similar.

- Env_BCC_Cert - value of this attribute should be comma separated string representing array of certificates (.cer files) which correspond to recipients marked as 'BCC' recipients (recipients given by 'bcc_addresses' attribute). For more information read about Env_TO_Cert argument, whose functionality is quite similar.

- Env_BCC_Keystore - value of this attribute should be comma separated string representing array of 'Key Stores' files which correspond to recipients marked as 'BCC' recipients (recipients given by 'bcc_addresses' attribute). For more information read about Env_TO_Keystore argument, whose functionality is quite similar.

- Env_BCC_KeystoreType - value of this attribute should be comma separated string representing array of 'Key Store' types, which correspond to recipients marked as 'BCC' recipients (recipients given by 'bcc_addresses' attribute). For more information read about Env_TO_KeystoreType argument, whose functionality is quite similar.

- Env_BCC_KeystorePassword - value of this attribute should be comma separated string representing array of 'Key Store' passwords, which correspond to recipients marked as 'BCC' recipients (recipients given by 'bcc_addresses' attribute). For more information read about Env_TO_KeystorePassword argument, whose functionality is quite similar.

- Env_BCC_KeystoreCertAlias - value of this attribute should be comma separated string representing array of 'Key Store' certificate aliases which correspond to recipients marked as 'BCC' recipients (recipients given by 'bcc_addresses' attribute). For more information read about Env_TO_KeystoreCertAlias argument, whose functionality is quite similar.

- Env_Algorithm - value of this attribute is string representing symetric algorithm type which will be used in enveloping (encription) of messages. The allowable values are DES, DES_EDE3_CBC, RC2_CBC. The chosen algorithm will be used for all recipients. If this argument is missing, the default definition, placed in Shark configuration file, is used (parameter SMIMEMailMessageHandler.Env.Default.Algorithm).

- Env_KeyLength - value of this attribute is string representing symetric algorithm key length which will be used in enveloping (encription) of messages. The allowable values for corresponding algorithms are: 56 (DES), 128,192 (DES_EDE3_CBC), 40, 64, 128 (RC2_CBC). If this argument is missing, the default definition, placed in Shark configuration file, is used (parameter SMIMEMailMessageHandler.Env.Default.KeyLength).

- Sig_Pfx - value of this attribute should be comma separated string representing array of certificates with private key (.pfx or .p12 files) which correspond to signers. Private keys and certificates are used for signing of message. The pfx files can be represented by their absolute paths, by their relative paths, or by their names only. In last two cases the default path from configuration file (parameter 'SMIMEMailMessageHandler.Sig.Default.Path') will be added as prefix to 'Private Key' files. The combination of all of this certificate definitions can be used as array items. Note that each member of array represents one signer. There are two parallel ways to define signer's private keys (via path to .pfx/.p12 files and from defined 'Key Stores').

- Sig_Pfx_Password - value of this attribute should be comma separated string representing passwords for access to corresponding .pfx/.p12 files. Note that number of items in this array should be equal to number of items in 'Sig_Pfx' array.

- Sig_Pfx_Algorithm - value of this attribute should be comma separated string representing signing algorithms used in process of message signing. Allowable values are: MD2_WITH_RSA, MD5_WITH_RSA, SHA1_WITH_DSA, SHA1_WITH_RSA. This algorithm depends on private key which is used for signing. For example, if private key is for RSA algorithm, then only combination of signing algorithms that relay on RSA can be used. Note that number of items in this array should be equal to number of items in 'Sig_Pfx' array. If any of items in array is missing (defined as empty - items with one or more space characters), the default value from Shark configuration file will be used (parameter SMIMEMailMessageHandler.Sig.Default.Algorithm).

- Sig_Pfx_IncludeCert - value of this attribute should be comma separated string representing decision whether to include or not certificate chain for particular signer (particular private key). Allowable array item values are: False and True. Note that number of items in this array should be equal to number of items in 'Sig_Pfx' array. If any of items in array is missing (defined as empty - items with

one or more space characters), the default value from Shark configuration file will be used (parameter SMIMEMailMessageHandler.Sig.Default.IncludeCert).

- Sig_Pfx_IncludeSignAttrib - value of this attribute should be comma separated string representing decision whether to include or not signed attributes for particular signer (particular private key). Allowable array item values are: False or True. Note that number of items in this array should be equal to number of items in 'Sig_Pfx' array. If any of items in array is missing (defined as empty - items with one or more space characters), the default value from Shark configuration file will be used (parameter SMIMEMailMessageHandler.Sig.Default.IncludeCert).

- Sig_Keystore - value of this attribute should be comma separated string representing array of java 'Key Stores' files with private key which correspond to signers. Private keys and certificates are used for signing of message. The 'Key Stores' can be represented by their absolute paths, by their relative paths, or by their names only. In last two cases the default path from configuration file (parameter 'SMIMEMailMessageHandler.Sig.Default.Path') will be added as prefix to 'Private Key' files. The combination of all of this certificate definitions can be used as array items. Note that each member of array represents the one signer. There are two parallel ways to define signer's private keyes (via path to .pfx/.p12 files mentioned above and from defined 'Key Stores').

- Sig_KeystoreType - value of this attribute should be comma separated string representing array of 'Key Store' types, which correspond to signers. Allowable values are: BKS, JKS, PKCS12, UBER. Note that number of items in this array should be equal to number of items in 'Sig_Keystore' array attribute. If any of items is missing, it should be defined as empty item in array (item with one or more space characters). Missing 'Key Store' types then must be found from default type definition placed in Shark configuration file (parameter SMIMEMailMessageHandler.Sig.Default.KeystoreType).

- Sig_KeystorePassword - value of this attribute should be comma separated string representing passwords for access to corresponding 'Key Store'. Note that number of items in this array should be equal to number of items in 'Sig_Keystore' array. If any of items is missing, it should be defined as empty item in array (item with one or more space characters). Missing 'Key Store' passwords then must be found from default password definition placed in Shark configuration file (parameter SMIMEMailMessageHandler.Sig.Default.KeystorePassword).

- Sig_KeystoreCertAlias - value of this attribute should be comma separated string representing array of 'Key Store' private key aliases which correspond to signers . Aliases are used to find desired private key from 'KeyStore'. Note that number of array items must be equal to number of 'Sig_Keystore' array items.

- Sig_Keystore_Algorithm - value of this attribute should be comma separated string representing signing algorithms used in process of message signing. Allowable values are: MD2_WITH_RSA, MD5_WITH_RSA, SHA1_WITH_DSA, SHA1_WITH_RSA. This algorithm depends on private key which is used for signing. For example, if private key is for RSA algorithm, then, only combination of signing algorithms that relay on RSA can be used. Note that number of items in this array should be equal to number of items in 'Sig_Keystore' array. If any of items in array is missing (defined as empty - items with one or more space characters), the default value from Shark configuration file will be used (parameter SMIMEMailMessageHandler.Sig.Default.Algorithm).

- Sig_Keystore_IncludeCert - value of this attribute should be comma separated string representing decision to whether to include or not signed attributes for particular signer (particular private key). Allowable array item values are: False or True. Note that number of items in this array should be equal to number of items in 'Sig_Keystore' array. If any of items in array is missing (defined as empty - items with one or more space characters), the default value from Shark configuration file will be used (parameter SMIMEMailMessageHandler.Sig.Default.IncludeCert).

- Sig_Keystore_IncludeSignAttrib - value of this attribute should be comma separated string representing decision to include or not to include signed attributesfor particular signer (particular private key). Allowable array item values are: False or True. Note that number of items in this array should be equal to number of items in 'Sig_Keystore' array. If any of items in array is missing (defined as empty - items with one or more space characters), the default value from Shark configuration file will be used (parameter SMIMEMailMessageHandler.Sig.Default.IncludeCert).

- Sig_ExternalSignature - value of this attribute should be string representing decision to make external or internal signature. Allowable values are: False or True. If this argument is missing, the default value from Shark configuration file will be used (parameter SMIMEMailMessageHandler.Sig.Default.ExternalSignature).

- Sig_CapabilSymetric - value of this attribute should be comma separated string representing symetric SMIME capabilities. Allowable values are: DES, DES_EDE3_CBC, RC2_CBC_40, RC2_CBC_64, RC2_CBC_128. If this argument is ommited, this capabilities information won't be included as one of signing information.

- Sig_CapabilEncipher - value of this attribute should be comma separated string representing ecipher SMIME capabilities. Allowable values are: RSA. If this argument is ommited, this capabilities information won't be included as one of signing information.

- Sig_CapabilSignature - value of this attribute should be comma separated string representing signature SMIME capabilities. Allowable values are: MD2_WITH_RSA, MD5_WITH_RSA, SHA1_WITH_RSA, SHA1_WITH_DSA. If this argument is ommited, this capabilities information won't be included as one of signing information.

Note that SMIME possibility should not be used until the original JCE Policy jar files are swapped with Unlimited Strength Java(TM) Cryptography Extension (JCE) Policy Files. The original JDK JCE Policy jar files, are located in JDK under the directory: **<jdk_home>/jre/lib/security**. In JRE, the original JDK JCE Policy jar files, are located under directory: **<jre_home>//lib/security**.The Unlimited Strength Policy files are shipped with this release, and can be found in directory: **.dist/crypto**.

- SchedulerToolAgent - proxy for calling other tool agents executed in separate thread.

  If you define XPDL automatic (tool agent) activity to have AUTOMATIC start, and MANUAL finish mode, shark kernel won't finish such activity automatically, but it will be the responsibility of the Tool Agent, or client application to do so. This approach can be used to start some application in a separate thread of execution, and SchedulerToolAgent is a right solution to do it easily.

  This tool agent takes responsibility to start other tool agent(s), and to finish corresponding activity when all of them finished their execution (which is in separate threads).

  This tool agent need one additional extended attribute to be defined:

  - ToolAgentClassProxy - value of this attribute should represent the full class name of the actual tool agent that should be executed in a separate thread.

  You may define other extended attributes that will be used by a actual tool agent which class name is defined in this attribute. I.e., if you want to use JavaScriptToolAgent, you might want to define "Script" extended attribute.

  The tool agent will provide all the parameters it gets (instance variables described by the formal parameters defined in the XPDL application definition) to the underlying tool agent.

Tool agents will read extended attributes only if they are called through "Default tool agent" (not by shark directly) and this is the case when information on which tool agent to start for XPDL application definition is not contained in mappings.

- DefaultToolAgent - this tool agent is called by shark when there is no mapping information for XPDL application definition. Its responsibility is to read extended attributes, try to find out the extended attribute whose name is "ToolAgentClass", read its value, and call appropriate tool agent determinated by the value of this extended attribute.

  All the parameters it gets (instance variables described by the formal parameters defined in the XPDL application definition) will be provided to the actual tool agent that will be executed.

  One can write the custom implementation of this tool agent, and he has to configure shark to use it, by changing configuration entry called **DefaultToolAgent**

- ToolAgentLoader - this is not actually a tool agent, but utility that is used to add new tool agents to the system while shark engine is running. You have to define the property called "ToolAgentPluginDir", and if shark can't find specified tool agent in the class path, this location will be searched for it's definition (in other words, put the jar file of your new tool agent into this folder and it will be recognized in the runtime).

How to Use Admin Application to Perform Tool Agent Mappings

You can map package and package's processes applications to the real applications handled by some tool agents. Currently, six agents (plus default tool agent) come with the Shark distribution.

To map application definition to tool agent application, you must have either demo or professional version of admin application. If so, you have to go to the application mapping section of admin application, and press the "add" button. The dialog will arise, and you have to select the application definition at the left side of dialog, and the tool agent on the right side of the dialog. Then you should enter some mapping parameters for tool agent:

- username and password - not required for tool agents distributed with Shark. Some other tool agents can use it when calling applications that require login procedure

- Application name - the name of application that should be started by tool agent (i.e. for JavaClassToolAgent that would be the full name of the class, for RuntimeApplicationToolAgent it would be the name of executable file that should be in the path of the machine where tool agent resides, for JavaScriptToolAgent and BshToolAgent this can be either the name of the java script file, or the java script itself, depending on Application mode attribute, for SOAPTtoolAgent it would be the location of WSDL file that defines web service, and for MailToolAgent it would be the name of MailMessageHandler interface implementation)

- Application mode - various tool agents use this attribute for different purposes. I.e. RuntimeApplicationToolAgent use mode "0" (zero) to indicate that it should wait until the system application is finished (otherwise it will start system application and finish its execution -> activity does not wait for system application to finish, but process proceeds to the next activity), JavaScriptToolAgent and BshToolAgent uses mode 0 (zero) to indicate that it should search for script file (otherwise, the application name will be considered to be the script to execute), and MailToolAgent uses mode 0 to indicate that mails will be send, and 1 if they should be received.

Example of Tool Agent Used in the Example XPDLs

If you load test-JavaScript.xpdl (or test-BeanShell.xpdl) by using Admin application, you can find out how Tool agents work.

This XPDL have defined extended attributes for application definitions, and these attributes contain data needed to call appropriate tool agents without need for mapping information (these tool agents are called through default tool agent by reading extended attribute parameters). The only thing you should do before starting shark engine is to configure your "Shark.conf" file to define proper settings for DefaultMailMessageHandler, but even if you don't do that, the example will work, because on MailToolAgent failure, it will proceed with DEFAULT_EXCEPTION transition.

If you want to do your own mappings, it will override default configuration in application's XPDL extended attributes because shark first looks at mapping information, and only if it can't find it, it calls Default tool agent, which reads ext. attributes. You can do the following to see how the mappings work:

- Start SharkAdmin application and go to the application mapping section
- Map the following:
  - addition -> org.enhydra.shark.JavaClassToolAgent (enter AdditionProc for application name)
  - arbitrarymathop -> org.enhydra.shark.SOAPToolAgent (enter http://samples.gotdotnet.com/quickstart/aspplus/samples/services/MathService/VB/MathService.asmx?WSDL for application name)
  - division -> org.enhydra.shark.JavaScriptToolAgent (enter c=a/b for application name, and 1 for application mode)
  - multiplication -> org.enhydra.shark.BshToolAgent (enter c=new Long(a.longValue()*b.longValue()); for application name, and 1 for application mode)
  - notepad -> org.enhydra.shark.RuntimeApplicationToolAgent (enter notepad for application name, and 1 for application mode) - this application is executed if shark is running on Windows
  - send_mail -> org.enhydra.shark.JavaClassToolAgent (enter email.MailProc for application name)
  - send_mail2 -> org.enhydra.shark.MailToolAgent (enter org.enhydra.shark.toolagent.DefaultMailMessageHandler for application name, and 0 for application mode)
  - substraction -> org.enhydra.shark.JavaScriptToolAgent (enter SubstractionProc.js for application name and 0 for application mode)
  - vi -> org.enhydra.shark.RuntimeApplicationToolAgent (enter xterm - e vi for application name, and 1 for application mode) - this application is executed if shark is running on *nix
  - waiting -> org.enhydra.shark.JavaClassToolAgent (enter WaitProc for application name)
- Instantiate the "Do math" process
- execute the given workitems (below is the explanation of the process)

The process is consisted of two loops:

- The first loop performs math operations using sub-process referenced from subflow activity "Calculate". When you perform "Enter math parameters" activity, enter some parameters, i.e. "addition", "44" and

"33", and when the subflow activity "Calculate" finishes, you should see the result of the addition ("77") when performing next activity. Here you can decide if you're going to repeat the calculation. If you are not, the process goes to the last activity, but it can't finish until the second loop is exited (you can also enter "substraction", "multiplication" and "division" for a operation parameter).

- The second loop is more interesting - it performs two operations:
  - it executes arbitrary mathematical operation
  - it executes waiting procedure

  Both operations have to be finished to continue the loop. Arbitrary mathematical operation is executed by calling WEB service, and waiting procedure uses java's sleep method.

  I.e., if you enter parameters "Add", "100.3","10.2","10000", the result of the arbitrary math operation that you will see when all operations are finished (if mapped as above) will be "110.5". You will be able to perform the activity that shows you the result of math operation, and asks you if you want to proceed with this loop, only when 10 second has past (you can also enter "Subtract", "Multiply" and "Divide" for a arbitraryOperation parameter - be careful, there is a typo in WSDL definition, so you should really enter "Subtract" if you want substraction to be performed).

When you decide to exit both loops, the process goes to "Notepad" or "Vi editor" activity, depending on OS that engine runs on, and appropriate text editor will be started on shark machine using RuntimeApplication tool agent, but process will continue to "Enter Mail Params" activity, because mode of RuntimeApplication tool agent is previously (in mappings) set to 1, which means asynchronous execution of editor application.

Now, you should enter some parameters to send e-mail notification to someone. I.e., enter something like this:

- txt_msg -> Do math process is finished
- subject -> Do math process status
- destination_address -> shark@enhydra.org

After that, the mail should be sent using MailToolAgent, and process will finish. If this is not the case, it means that you didn't setup appropriate parameters in Shark.conf file, so exception in tool agent will happen, but since XPDL has defined DEFAULT_EXCEPTION transition, the process will proceed to exception handling path -> to activity "Enter Aditional Mail Params". Now, you should enter additional parameters that are needed by email.MailProc class used to send mails through JavaClass tool agent. I.e., enter something like this:

- source_address -> admin@together.at
- user_auth -> admin
- pwd_auth -> mypassword
- server -> myserver
- port -> 25

After that, process will be finished no matter if you've entered proper parameters or not.

Now, you can play around with the mappings. I.e., you can enter different java script text for executing math operations, enter different parameter values, ...