
Enhydra Shark Tool Agents

Table of Contents

About tool agents (quotation from WfMC document)	1
Shark Implementation of Tool Agent Interface	1
How does Shark Use Tool Agents	2
Shark Tool Agent Examples	3
How to Use Admin Application to Perform Tool Agent Mappings	6
Example of Tool Agent Used in the Example XPDLs	6

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](http://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 some automatic* activity of "Tool" type is performed, shark searches for the appropriate tool agent:

- if mapping information for this activity's application definition 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 it 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 XPD application, chopped out from the XPD definition, i.e.:

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

As previously mentioned, if shark can't find 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 XPD 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 activity is "Tool" type activity, and its performer is "SYSTEM" type participant, and activity's Start mode is AUTOMATIC
2. If activity is "Tool" type activity, and its performer is an empty expression (it doesn't have performer) and activity's Start mode is AUTOMATIC
3. If activity is "Tool" type activity, and it has a performer different then "SYSTEM" type participant, and its Start mode is MANUAL

If the last one is the case, the activity will first be assigned to participant, and after it finishes it, the tools specified will be executed automatically by the shark, through tool agent calls.

Shark Tool Agent Examples

We have developed six useful tool agents (plus default tool agent) to give you an example of 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
- **RuntimeApplicationToolAgent** - Executes some system applications like notepad or any other executable application. It is important that this application should 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 than 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 starting of some external applications).

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

This tool agent is able to "understand" the extended attributes with the 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.
- **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, 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** - 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 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)

- 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. So far, we didn't define any script files.

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

- AppName - 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)

- 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.

Also, 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.

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.
- MailToolAgent - sends and receives mail messages.

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

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
```

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.
- `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.
- `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).

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

```
# Credentials for Scheduler Tool Agent
SchedulerToolAgent.sharkUsername=admin
SchedulerToolAgent.sharkPassword=enhydra
```

This tool agent need one 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.

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 determined in the value of this extended attribute.

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.

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 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 SOAPToolAgent 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)
 - subtraction -> 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 subtraction 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 Additional 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, ...