Witmate Engine
Users Manual

Version 603

mate of your wit

Contents

Introduction

Witmate engine is a pure java logic engine, which is called as Rule engine in some application areas as CRM (Customer Relation Management), for wide usages from mobile devices logic execution to enterprise high-end business rule matching.

Delivering their business experts logic into IT system quickly and updating these logics time to time are an important approach to keep their competitions for enterprises. And it is a new challenge to enterprises' IT support teams. Witmate engine is the solution to solve the hard problem for enterprise and their IT infrastructures.

As a widely usable logic engine, its application areas are not limited as talked above. From general point of view, Witmate engine is a development tool to help application fields' experts and programmers focusing on their own fields and cooperating together smoothly.

Features

- Mobile/Home network environment ready

Witmate engine is a mobile environment ready logic engine. "Mobile ready" is not only figure out that it is executable on J2ME and OSGi platform, but also support architecture to create, manage and update logics through mobile networks.

Witmate engine exposes a JSR-94 like API on J2ME platform. This makes it easy to integrate mobile devices into enterprise system, and lower down the cost to manage and develop systems on both of mobile and typical enterprise platform.

- Web services/SOA ready

All components of Witmate engine are SOAP ready. So it's very easy to deploy Witmate engine into Web services/SOA systems.

- JSR-94 compatibility

Witmate engine is compatible with JSR-94, the standard of rule engine of java. Witmate engine is the only one engine in the world that supports both of J2ME and JSR-94. There is a API that is very close with JSR-94 in Witmate core engine, so experiences of JSR-94 can be used easily to code with core engine in mobile environment, and size up smoothly from mobile to enterprise environment at anytime you need.

A JSR-94 Witmate engine cover layer implemented the standard JSR-94 in J2SE environment for enterprise applications. This layer passed all test cases of JSR-94 TCK (Java Rule Engine API Test Compatibility Kit).

- Multiple logic description formats

Not like other engines that offer only one logic description language to users to create logics,

as another leading advantage, Witmate engine offers 3 description formats to users for different targets and environments. They are:

1. Simple Logic Language (SL): An easily understandable, human friendly language to create logics. Without strange expressions as Prolog or other logic expression languages, Witmate engine just uses IF-THEN to express logic. Anybody can start to use in minutes. SL also offers other expressions as vocabulary definitions and variables definitions to make the communications between IT team and field experts more smoothly.
   This is the best format to create and modify logics by experts themselves or together with IT team.
2. Simple Logic Markup Language (SLML): A XML format of logics. SLML is designed to get balance of readability and compact XML string size, so it is not cost to send SLML through a mobile network.
   This is the best format to create/manage logics by computers, and exchange logics among computers or software components.
3. WitStream (WITS/WS): A text based stream format of logics. This is Witmate engine special format; no another engine offers this type format of logics. This format is very compact, normally one of third of length of XML format. As a very compact text stream, WitStream can be exchanged through any mobile or fixed networks, can be saved into any storages as Data Bases, Files or stick memories, or even printed on paper as back up (Paper back up still is the last approach for very important data as forms of insurances).
   WitStream is the best format to exchange/broad cast logics through mobile networks, and save/retrieve logics to/from database or any storage.

- Logic Set Firing

Logic sets in Witmate engine can fire other logic sets at runtime. This makes it easier to describe complex logics.

With logic set firing, Witmate engine can execute logics as status machine to solute much more complex problems than normal single logic set approach.

- High performance

Leveraging advantaged system structure and logic optimization, Witmate engine offers high performance on mobile and enterprise environment. Over 200 thousands compares and processes can be done in one second by Witmate engine even on a typical thin PC (1GHz CPU, 1G RAM).

Concerning the fact that any cache system has its limit for certain environment, Witmate engine has an optional cache component. It is possible to get high performance for any environments by configuring and open/close Witmate engine cache.

- Small footprint

Very small memory and CPU usage makes Witmate engine can execute on very weak mobile devices.

- Scalability

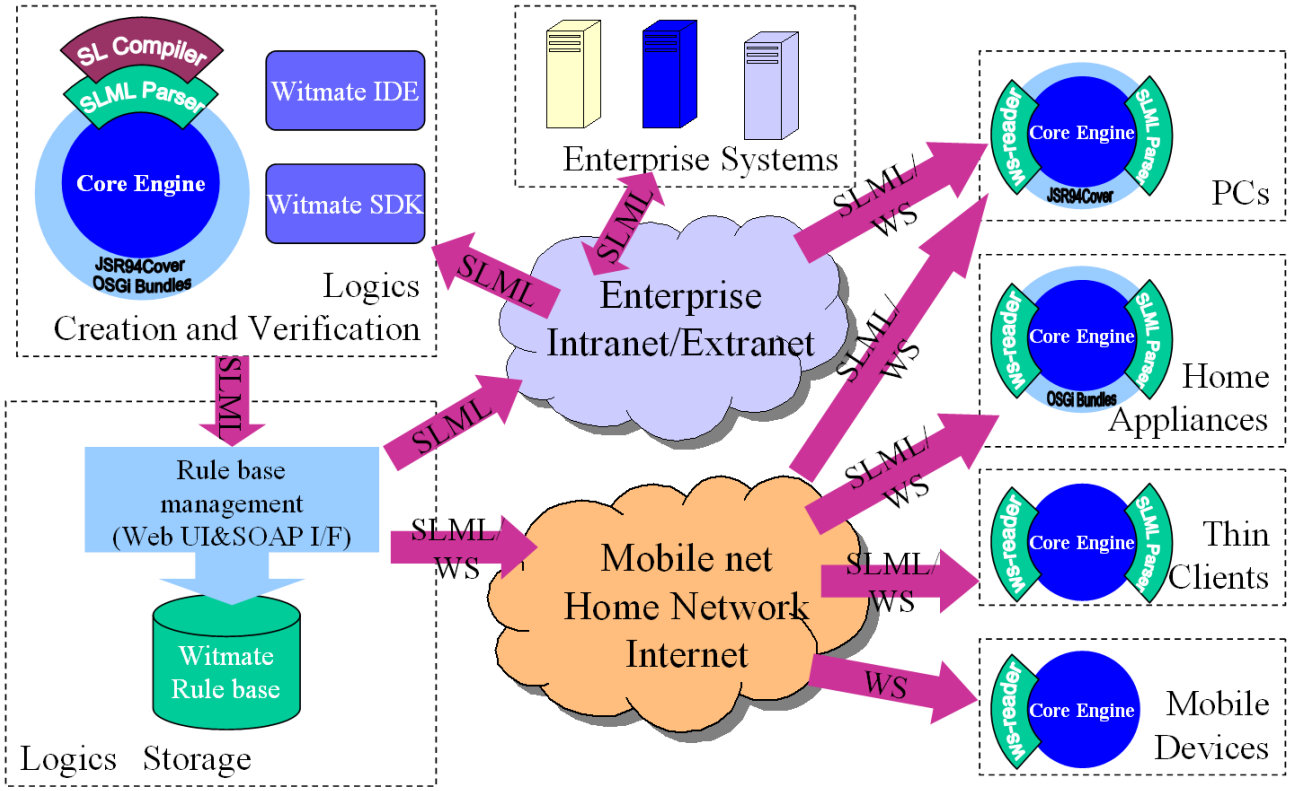
Witmate engine system is separated into several JAR files; so different target applications only deploy the parts they need.

To deploy Witmate engine easily on enterprise environment, an all in one JAR file is offered.

Typical Applications

Mobile logic engine and logic update/broadcast system

As shown below, Witmate engine can be used to manage and execute logics in an Enterprise + Mobile + Home network environment.



- Core Engine with JSR-94 Cover plus SLML parser and SL compiler construct a logics creation and verification subsystem. Experts/Programmers create logics by SL, and load logics into engine by SL compiler and SLML parser through standard JSr-94 interface. Then executing logics in core engine verify these logics.
- Verified logics will be load into logic storage subsystem that is construed by Core Engine with JSR-94 cover and SLML Parser, WS reader, WS writer by SLML format, and be saved into storages by WitStream format.
- An exchange and broadcast subsystem exchange logics with other enterprise servers, and broadcast logics into clients from PC to Mobile devices, using SLML and WitStream. According to Clients power, deploy different components (JARs) into different type clients.

Business Rule engine

This is the typical usage of JSR-94 compatible engines. For example a CRM (Customer Relation Management) rule engine decides what services should be served to the concurrent

accessing customer according to this customer's profiles as input.

Application Logic creation/management/execution system

This is a very wide usage of Witmate engine. Because of high performance, small footprint and multiple logic formats, Witmate engine can be used as logic engine in any applications, from mobile gaming, Web site, or even data reformat programs. Basically, any situation there are requirements of communicating logics with other members or changing logics separated with codes.

Typical steps to use Witmate engine

1. Create logics by Simple Logic language (SL) with an editor
2. Load SL created into engine with SL compiler.
3. Test/Debug with WitDebugger.
4. (Optional) Export logics to SLML using sl2slml tool, and send logics to other systems with SOAP/Web services
5. (Optional) Export logics to WitStream with core engine API or slml2ws tool, and save logics into Witmate Rule base (Refer Witmate rule base users manual).
6. Deploy Witmate engine with required components into execution environment, and use it!

Configuration

Witmate engine is a total configurable logic engine to fit different environments and application requirements. For most environments, default values of configurations are work well. To deal with very special requirements, follows approaches below to configure Witmate engine.

There are tow ways to configure Witmate engine, changing values of keys in Witmate.core.Configuration class, or using Witmate engine Witmate.core.ConfigManager.configure method to load configurations from a stream (File, mobile network stream or any input can be read as InputStream).

For details of configuration keys, refer java doc of Witmate.core.Configuration and Witmate.properties file.

Application Program Interface (API)

WitME API

WitME API is a JSR-94 like API exposed by Witmate core engine. It is executable on J2ME and any JRE/JDK.

Because the WitME API is so similitude with JSR-94, the JSR-94 documents (http://www.jcp.org/aboutJava/communityprocess/review/jsr094/) and Witmate engine JavaDoc include enough information about usage of WitME API.

1. Differentiations between WitME and JSR-94, Since limits of J2ME,
   - Change all Map parameters to Hashtable
   - Change all List parameters to Vector
   - Can't set ClassLoader in RuleServiceProvider
   - Does NOT support the method createRuleExecutionSet that has AST parameter in LocalRuleExecutionSetProvider
   - Deleted the class RuleExecutionSetProvider
2. Limit of Witmate engine
   - Do not support retrieve Rules from RuleExecutionSet/RuleSession
3. Extension of Witmate engine
   - Extend methods in LocalRuleExecutionSetProvider to get an array of RuleExecutionSet to load more than one logic set into engine at one time.
   - Extend method in RuleAdministrator to deregister one RuleExecutionSet from one URI binding.
   - Extend a lightweight session provider to simple client codes. One call to LightProvider. createRuleSession replaces all callings from registerRuleServiceProvider to createRuleSession of standard JSR94. Example:

```
RuleSession rs=LightProvider.createRuleSession(uri, new Hashtable(), input_stream,
                    RuleRuntime.STATELESS_SESSION_TYPE);
Vector result = ((StatelessRuleSession)rs).executeRules(m_vInVars);
rs.release();
LightProvider.deregisterRuleExecutionSet(uri,m_hProp);
```

4. Start logic set decision

   Witmate engine can bind more than one logic set to a URI, the logic set to start match is decided by:

   1. Look for Wit-Start-SLName property in property parameter of RuleAdministrator. createRuleSession method
   2. If there is not this property, and more then one logic set are loaded, look for logic set name by these steps:
      I. Equals “start” (insensible)
      II. Equals “begin” (insensible)

III. Equals “main” (insensible)
IV. Starts with “start” (insensible)
V. Starts with “begin” (insensible)
VI. Starts with “main” (insensible)
VII. Includes “start” (insensible)
VIII. Includes “begin” (insensible)
IX. Includes “main” (insensible)

3. If cannot found, uses the first logic set loaded.

1. Input format decision

Because Witmate engine has 3 formats of logics, it uses these steps to decide input format:

1. Look for Wit-SL-InType property in createRuleExecutionSet methods of LocalRuleExecutionSetProvider. Value should be String “1”, “2” or “3”.

1	Compile as Simple Logic language (SL)
2	Parse as Simple Logic Markup Language (SLML)
3	Read as WitStream

2. If no this property, Witmate engine try to decide input type by contents of input

Begin with <?xml…	Parse as Simple Logic Markup Language (SLML)
Begin with WITS	Read as WitStream
Other situations	Compile as Simple Logic language (SL)

2. Runtime input

Witmate engine support 2 ways to input at runtime in both of stateless and stateful session.

1. Input any instances of class. This is the JSR-94 approach. The input values will be assigned into variables which has same class type defined by Def command in SL or c attribute of SLML..
2. Input instances of Witmate engine .core.kernel.elements.ConstantVariable. This is extension of Witmate engine. Inputs and variables in logic set linked by variable name, so it is possible that input more than one same class inputs.

3. Results output

Witmate engine support 2 ways to return results at runtime in both of stateless and stateful session.

1. Output instances of result values. This is JSR-94 approach. Client code need identify results by sort or class of results.
2. Output instances of Witmate engine .core.kernel.elements.Constant or Witmate engine .core.kernel.elements.ConstantVariable. This way makes identify results easier. Refer “Return/ReturnC” command segment for details.

4. Important WitME classes: These are some important class of WitME, refer their

JavaDoc for details.

- Witmate engine .core.ConstantValues: This class defines many constant values to help client coding.
- Witmate.core.Configuration: This class defines all configurations of Witmate engine, change them to configure Witmate engine at runtime.

5. Multiple thread support

   To support multiple thread environments, threads need to:

   - Generate separated execution session for each thread
   - Manage URI of execution set(s)
     - Using unique URI, if want to separate logic sets of threads
     - Figuring out certain URI for certain thread logics.

Examples

This is a test case of Witmate core engine with LS logics format.

```
String RULE_SERVICE_PROVIDER = "Witmate engine ";
Class provider=Class.forName( "Witmate engine .core.kernel.ServiceProvider" );
//Get the rule service provider from the provider manager.
RuleServiceProviderManager.registerRuleServiceProvider(RULE_SERVICE_PROVIDER,provider);
m_serviceProvider= RuleServiceProviderManager.getRuleServiceProvider(RULE_SERVICE_PROVIDER);
//get the RuleAdministrator
m_ruleAdministrator = m_serviceProvider.getRuleAdministrator();
String test="#This is a simple test LS file for with operators ¥n" +
            "LogicSet LSetMainOpSlT1 ; #a simple logic set.¥n" +
            "           Def Integer V1,V2; Let V1 = 10 + 20 - 5¥n" +
            "           IF V1>3 THEN Publish V1; Let V2=V1*2; ReturnC V2 ";
ByteArrayInputStream bis=new ByteArrayInputStream(test.getBytes());
ByteArrayOutputStream bsOut=new ByteArrayOutputStream();
OutputStreamWriter streamOut=new OutputStreamWriter(bsOut);
Hashtable prop=new Hashtable();
prop.put(ConstantValues.SLML_OUT_KEY, streamOut);
//parse the logic set
RuleExecutionSet res=
m_ruleAdministrator.getLocalRuleExecutionSetProvider( null ).createRuleExecutionSet( bis, prop );
System.out.println(bsOut.toString());//Print out SLML.
bis.close();
//register the RuleExecutionSet with an unique URI.
m_ruleAdministrator.registerRuleExecutionSet(Thread.currentThread().hashCode(), res, null );
//Get a RuleRuntime and invoke the rule engine.
RuleRuntime ruleRuntime = m_serviceProvider.getRuleRuntime();
//create a StatelessRuleSession
StatelessRuleSession statelessRuleSession = (StatelessRuleSession) ruleRuntime.createRuleSession(uri,
      new Hashtable(), RuleRuntime.STATELESS_SESSION_TYPE);
//Execute the rules without a filter.
Vector results = statelessRuleSession.executeRules(null);
//Release the session.
statelessRuleSession.release();
TestCase.assertEquals(1, results.size());
TestCase.assertEquals(new Integer(50), ((ConstantVariable)results.elementAt(0)).value);
TestCase.assertEquals("V2", ((ConstantVariable)results.elementAt(0)).name);
TestCase.assertEquals(Witmate engine .core.ConstantValues.DT_INT,
        ((ConstantVariable)results.elementAt(0)).type);
```

Using lightweight session provider to get a simpler client code:

```
String test="#This is a simple test LS file for with operators ¥n" +
            "LogicSet LSetMainOpSlT1 ; #a simple logic set.¥n" +
            "           Def Integer V1,V2; Let V1 = 10 + 20 - 5¥n" +
            "           IF V1>3 THEN Publish V1; Let V2=V1*2; ReturnC V2 ";
ByteArrayInputStream bis=new ByteArrayInputStream(test.getBytes());
ByteArrayOutputStream bsOut=new ByteArrayOutputStream();
OutputStreamWriter streamOut=new OutputStreamWriter(bsOut);
Hashtable prop=new Hashtable();
prop.put(ConstantValues.SLML_OUT_KEY, streamOut);
StatelessRuleSession statelessRuleSession=
    (StatelessRuleSession)LightProvider.createRuleSession(Thread.currentThread().hashCode(), prop, bis,
    RuleRuntime.STATELESS_SESSION_TYPE);
System.out.println(bsOut.toString());//Print out SLML.
bis.close();
//Execute the rules without a filter.
Vector results = statelessRuleSession.executeRules(null);
//Release the session.
statelessRuleSession.release(); ….
```

JSR-94 API

WitPeel.jar implements JSR-94 standard, and passed the JSR-94 TCK. Refer JSR-94 document (http://www.jcp.org/aboutJava/communityprocess/review/jsr094/) and Witmate engine JavaDoc for usages.

Differentiations between WitPeel and JSR-94

1. Limits of Witmate engine
   - Do NOT support methods createRuleExecutionSet that has AST parameter and URI paramater in RuleExecutionSetProvider
   - Does NOT support the method createRuleExecutionSet that has AST parameter in LocalRuleExecutionSetProvider
2. Extension of Witmate engine
   - Extend methods in LocalRuleExecutionSetProvider to get an array of ExecutionSet to load logic sets one time into engine.
   - Extend methods in RuleAdministrator to regsiter/deregister by RuleExecutionSet or array of RuleExecutionSet.
3. Start logic set decision

   Witmate engine can bind more than one logic set to a URI, the logic set to start match is decided by:

   1. Look for Wit-Start-SLName property in property parameter of RuleAdministrator. createRuleSession method
   2. If there is not this property, and more then one logic set are loaded, look for logic set name which is:
      I. Equals or starts with “start” (insensible)
      II. Equals or starts with “begin” (insensible)
      III. Equals or starts with “main” (insensible)
      IV. Includes “start” (insensible)
      V. Includes “begin” (insensible)
      VI. Includes “main” (insensible)
   3. If cannot found, uses the first logic set loaded.
4. Input format decision

   Because Witmate engine has 3 formats of logics, it uses these steps to decide input format:

   - Look for Wit-SL-InType property in createRuleExecutionSet methods of LocalRuleExecutionSetProvider. Value should be String “1”, “2”, “3” or “4”.

1	Compile as Simple Logic language (SL)
2	Parse as Simple Logic Markup Language (SLML)
3	Read as WitStream

4	Parse as Simple Logic Markup Language (SLML) with slml validation

- If no this property, Witmate engine try to decide input type by contents of input

Begin with <?xml…	Parse as Simple Logic Markup Language (SLML)
Begin with WITS	Read as WitStream
Other situations	Compile as Simple Logic language (SL)

5. Runtime input

   Witmate engine support 2 ways to input at runtime in both of stateless and stateful session.

   1. Input any instances of class. This is the JSR-94 approach. The input values will be assigned into variables which has same class type defined by Def command in SL or c attribute of SLML.
   2. Input instances of Witmate engine .core.kernel.elements.ConstantVariable. This is extension of Witmate engine. Inputs and variables in logic set linked by variable name, so it is possible that input more than one same class inputs.

6. Results output

   Witmate engine support 2 ways to return results at runtime in both of stateless and stateful session.

   1. Output instances of result values. This is JSR-94 approach. Client code need identify results by sort or class of results.
   2. Output instances of Witmate.core.kernel.elements.Constant or Witmate.core.kernel.elements.ConstantVariable. This way makes identify results easier. Refer “Return/ReturnC” command segment for details.

7. Multiple thread support

   To support multiple thread environments, threads need to:

   - Generate separated execution session for each thread
   - Manage URI of execution set(s)
     - Using unique URI, if want to separate logic sets of threads
     - Figuring out certain URI for certain thread logics.

Examples

This is a test case of WitPeel with SL logic format.

```
String test="LogicSet LSetClassVarTest ResultCount 1¥n" +
            "           Def obj c,vector; Def int i; " +
            "           Def java.lang.Character char¥n" +
            "           Let c=char;"+
            "           :1:If i<0 OR c=char Then" +
            "           :5:do Vector$Add(vector,¥"5¥");:2:do Vector$Add(vector,¥"2¥")" +
            "           return c, char, vector";
ByteArrayInputStream bis=new ByteArrayInputStream(test.getBytes());
ByteArrayOutputStream bsOut=new ByteArrayOutputStream();
OutputStreamWriter streamOut=new OutputStreamWriter(bsOut);
prop.put(Configuration.KEY_SL_MAX_ERR_COUNT,"1");
//parse the ruleset
RuleServiceProvider serviceProvider = (RuleServiceProvider)Class.forName("Witmate
engine .core.kernel.ServiceProvider").newInstance();
RuleExecutionSet res =
serviceProvider.getRuleAdministrator().getLocalRuleExecutionSetProvider( null ).createRuleExecutionSet(
bis, prop );
bis.close();
//register the RuleExecutionSet
String uri = res.getName();
serviceProvider.getRuleAdministrator().registerRuleExecutionSet(uri, res, null );
//Get a RuleRuntime and invoke the rule engine.
RuleRuntime ruleRuntime = serviceProvider.getRuleRuntime();
Character ch=new Character('C');
ConstantVariable V1=new ConstantVariable();
V1.name="i";
V1.type=ConstantValues.DT_INT;
V1.value=new Integer(100);
ConstantVariable vector=new ConstantVariable();
vector.name="vector";
vector.type=ConstantValues.DT_OBJ;
vector.value=new Vector();
Vector in_vars=new Vector();
in_vars.addElement(ch);
in_vars.addElement(V1);
in_vars.addElement(vector);
//create a StatelessRuleSession
StatelessRuleSession statelessRuleSession = (StatelessRuleSession) ruleRuntime.createRuleSession(uri,
        new Hashtable(), RuleRuntime.STATELESS_SESSION_TYPE);
Vector results = statelessRuleSession.executeRules(in_vars);
//Release the session.
statelessRuleSession.release();
TestCase.assertEquals(3, results.size());
TestCase assertEquals(ch, results.elementAt(0));
TestCase.assertEquals(ch.charValue(), ((Character)results.elementAt(0)).charValue());
TestCase.assertEquals(ch, results.elementAt(1));
TestCase.assertEquals(ch.charValue(), ((Character)results.elementAt(1)).charValue());
TestCase.assertEquals(vector.value, results.elementAt(2));
TestCase.assertEquals("2", ((Vector)results.elementAt(2)).elementAt(0));
TestCase.assertEquals("5", ((Vector)results.elementAt(2)).elementAt(1));
```

SOAP Interface

This SOAP interface exposes Witmate engine functionalities to any Web services compatible systems. Current version only supports stateless rule sessions.

Configurations

There is a configuration file in root of classes' folder. Configurations include:

witmate_rulebase_endpoint

Set the rule base SOAP interface end point used by this Engine SOAP interface. This is a static way with same function of method *setWitmateRulebaseEndpoint.*

witmate_rulebase_synchronize

Set this Engine SOAP end point to synchronize with its rule base. This is a static way with same function of method *synchronizeRulebase.*

Methods

All methods are list below:

setWitmateRulebaseEndpoint

Definition

void setWitmateRulebaseEndpoint(String endpoint)

Parameters

endpoint: A string of URL to figure out the rule base SOAP interface end point used by this Engine SOAP interface

Function

Set the rule base SOAP interface end point used by this Engine SOAP interface

Example

your_proxy.setWitmateRulebaseEndpoint("http://localhost:8080/WitmateRulebase/Rules");

Comments

This method must be called before all other methods calls for most case besides of using the local configuration by file witmate.properties in class folder of Web SOAP application. If there is not any rule base installed in your system, you need not to call this method.

synchronizeRulebase

Definition

void synchronizeRulebase(String engine_endpoint)

Parameters

engine_endpoint: Set this Engine SOAP end point to synchronize with its rule base.

Function

Set this Engine SOAP end point to synchronize with its rule base.

Example

your_proxy. synchronizeRulebase
("http://localhost:8080/WitmateEngineWS/services/BasicRunner");

Comments

This method must be called before all other methods calls for most case besides of using the local configuration by file witmate.properties in class folder of Web SOAP application. If there is not any rule base installed in your system, you need not to call this method.

registerRuleExecutionSet

Definition

void registerRuleExecutionSet(String uri, Map properties, String in)

Parameters

uri: A string of URI execution set registered to.
properties: Register properties (Refer segment of Application Program Interface)
in: The logic/rule description by SL, SLML or WitStream

Function

Register logic/rule sets by an URI.

Example

your_proxy. registerRuleExecutionSet ("your.logic.logic1", new HashMap(0), your_logics);

Comments

For systems without Rule base, this method must be called before creating rule sessions. You need not call this method for a system with rule base. In a system with Rule base, this method has higher privilege than Rule base. This means that rule sets registered by this

method will not be searched in Rule base.

deregisterRuleExecutionSet

Definition

void deregisterRuleExecutionSet (String uri, Map properties)

Parameters

uri: A string of execution set URI.

properties: Deregister properties (Refer segment of Application Program Interface)

Function

Deregister logic/rule sets by an URI.

Example

your_proxy. deregisterRuleExecutionSet ("your.logic.logic1", new HashMap(0));

Comments

Deregister execution sets that are registered by certain URI. You need not call this method for a system with rule base. In a system with Rule base, this method will force engine to reload execution sets from Rule base when a new rule session with same URI is created.

createRuleSession

Definition

String createRuleSession(String uri, Map properties)

Parameters

uri: A string of execution set URI.

properties: Creation properties (Refer segment of Application Program Interface)

Return

Session key

Function

Create a stateless rule session of an URI.

Example

String session_key= your_proxy.createRuleSession("your.logic.logic1", new HashMap(0));

Comments

For a system with Rule base, this method will make engine to load logic/rule sets from Rule base.

For a system without Rule base, this method will throw exception, if registerRuleExecutionSet method is not called before.

releaseSession

Definition

void releaseSession(String session_key)

Parameters

session_key: rule session key created by method createRuleSession.

Function

Release a stateless rule session.

Example

your_proxy.releaseSession ("your.logic.logic1");

Comments

Release sessions that you do not use it in long term, to save system resources.

executeSession

Definition

Vector executeSession(String session_key, Variable[] input_vars)

Parameters

session_key: A rule session key created by method createRuleSession.

input_vars: Input variables that are instances of class Variable

Return

Results list, values or instances of Variable according to results are returned by Return or ReturnC command.

Function

Create a stateless rule session of an URI.

Example

Vector results=your_proxy.executeSession(session_key, vars);

Comments

This is the key method by which logic/rule sets are matched.

Typical using scenario

Systems with Rule base

1. Call setWitmateRulebaseEndpoint
2. Call createRuleSession to get session key
3. Call executeSession with session key to match rules
4. Process results
5. Recall executeSession with new input, if need
6. Call releaseSession with session key

Standalone without Rule base

1. Call registerRuleExecutionSet
2. Call createRuleSession to get session key
3. Call executeSession with session key to match rules
4. Process results
5. Recall executeSession with new input, if need
6. Call releaseSession with session key
7. Call deregisterRuleExecutionSet

REST Interface

This REST (Representational State Transfer) interface is a high performance replacer of SOAP. A REST interface speed up engine accesses from 30% to 60%.

Right now, REST interface does not support functions of SOAP as below:

5. Runtime rule RuleExecutionSet register
6. Rule base synchronization
7. Rule session variables
8. createRuleSession with properties

Configurations

REST interface can be configured through <init-param> tag in web.xml. Configurations include:

witmate_rulebase_endpoint

Set the rule base SOAP interface end point used by this Engine REST interface. This is a static way with same function of method *setWitmateRulebaseEndpoint.*

Methods

A simple methods list can be get by access root of REST web application as http://<Host name>:<port number>/WitmateEngineREST/. Same contents are list below:

setWitmateRulebaseEndpoint	p1=<End point of witmate rule base>	Set the SOAP end point of Witmate Rule base.	OK	Error messages
createRuleSession	uri=<Logic set URI>	Create a rule session of certain logicset	<rule session ID>	Error messages
executeSession	sid=<rule session ID> *vars*=<XML according to witmate_variable.xsd>	Execute a rule session with input variables	Results separated by line break	Error messages
attachGlobalVariables	uri=<Logic set URI> vars=<XML according	Attach global variables	OK	Error messages

	to witmate_variable.xsd>			
releaseSession	sid=<rule session ID>	Release a rule session	OK	Error messages
detachGlobalVariables	uri=<Logic set URI>	Detach global variables attached with certain logic set	OK	Error messages

* *Italic* : Optional

OSGi Bundles

OSGi is a standard of mobile, car and home networks (http://www.osgi.org/). Witmate engine supports OSGi standard too, and exposes bundles of Witmate engine. There are 2 approaches to use Witmate engine in OSGi system:

① Include Witmate engine jars which are needed into any bundles. Because Witmate engine works well on any OSGi environments, it's very easy to include any jars of Witmate engine as inside lib in bundles.

② Use Witmate bundles. There are 3 standard Witmate bundles in WitSDK:

1. WitCoreBundle: Include WitCoreAll.jar and SLMLParser.jar. Works on any OSGi environment.
2. WitSLCompilerBundle: Bundle of SLCompiler. Need this bundle to compile SL format input. Works on any OSGi environment.
3. WitLoggerBundle: Use OSGi log service to output Witmate engine log. Need this bundle to output log into OSGi system or other Bundles. Works on OSGi environment that installed log services. Witmate engine log level is debug, so the OSGi environment log level need set to debug to get log output.

There is an example Bundle in WitSDK.

A typical bundles install/start sequence to use Witmate bundles is:

Bundles	**Start Level**	**Mode**	**Comments**
Configuration management service	2	Start	Optional. Need for log.
Log service	2	Start	Optional. Need for log.
WitLoggerBundle	3	Start	Optional. Need for log.
WitSLCompilerBundle	5	Install	Optional. Need for SL logics input.
WitCoreBundle	6	Start	
Client Bundle using Witmate engine	7	Start/Install	

Log system

Log is the important tool to debug, trace and analysis logic sets and theirs executions. Witmate engine offers a total configurable log system based on Log4j2ME, a J2ME edition of Log4J, for j2ME and Log4j for J2SE/J2EE.

Log format

General Format

Witmate engine Logs follow a general format:

WITLOG(*information type*)[*LogicSet Name*]{*Type special format*}'EOL'

All Witmate engine Logs begin with WITLOG to make it easy to separate Witmate engine Logs with other Logs in one Log file. *information type* is a String to figure out what this Log is about. *LogicSet Name* is the name of LogicSet that logged out this Log. *Type special format* is changeable for each type of Log. EOL is the break of line.

All *information type* and their details refer every type of Log in Types of Logs segment below.

These are the formats of some standard Log elements

Variables

[*var_name*:*var_type*:*value*] or [*var_name*:?]

The first format is for variables that has value. The second is for non value variable or the Log type that need not Log out the value of variables.

var_name is the name of this variable.

var_type is a character about assignment type of this variable. The possible values are:

Character	Assignment type
c	This Variable is assigned by a constant. Like: Let a=1
r	This variable is assigned by referring to another Variable. Like: Let a=b
e	This variable is assigned by an expression. Like: Let a=1+b

Value is the values of this variable.

Constants

[*value*]

Value of constant

Expressions

[*operator*:*parameter list*]

operator is the operator full class name, *parameter list* is the list of parameters input to this

operator.

Examples:

```
WITLOG(LSet Start)[slmain]{[in_a:L][in_b:10]}
```

This Log figure out that a LogicSet named slmain started with inputs in_a with value 'L' and in_b with value 10.

Types of Logs

Witmate engine classify Logs into 4 types which can be open/close independently, they are:

8. Process Log (ProcLog)

Log out the process trace of LogicSets Start/End and important process:

Information type	**Meaning**	**Format(in one line)**
LSet Start	A LogicSet started	WITLOG(LSet Start) [*StartedLogicSetName*] {*InputVariableList*}
Fire LSet	Fired a LogicSet	WITLOG(Fire LSet) [*CurrentLogicSetName*] {*FiredLogicSetName InputVariableList* }
LSetReturn	Return from a LogicSet	WITLOG(LSetReturn) [*LogicSetName*]{*ReturnVariable/ValueList*}
Refer LSet	Refer a LogicSet	WITLOG(Refer LSet) [*LogicSetName*]{*ReferredLogicSetName*}
OutRefLSet	Got out from a referred LogicSet	WITLOG(OutRefLSet) [*LogicSetName*]{*ReferredLogicSetName*}
PublishVar	Published a variable into Public zone.	WITLOG(PublishVar) [*LogicSetName*]{*PublishedVariablesList*}
Then Start	A then command block are stared. Clarify what commands are executed by a then command block.	WITLOG(Then Start) [*LogicSetName*] {----------}
Else Start	An else command block are stared. Clarify what commands are executed by an else command block.	WITLOG(Else Start) [*LogicSetName*] {----------}

ThenFailed	Failed to execute a then/else command block	WITLOG(ThenFailed) [*LogicSetName*] {*ReasonOfFailure*}
PTime LS	Performance Log. How many time units are used for a Logic fire/refer or a logic execution.	WITLOG(PTime LS) [*LogicSetName*] {*Number* unit}
PTime All	Performance Log when a LogicSet execution finished. Log out total time unit used and time percentages of every LogicSet used. This is helpful to figure out which LogicSet should be focused on to rewrite to get better performance.	WITLOG(PTime All) [*LogicSetName*]{*TotalNumber* unit: *LogicSetsTimeUnitList*}

9. Compare Log

Log out all compares trace and result

Information type	**Meaning**	**Format (in one line besides ExceptionMessage)**
Cmpr Start	A compare, logic match, started	WITLOG(Cmpr Start) [*LogicSetName*] {----------}
CompareRlt	Result of a compare	WITLOG(CompareRlt) [*LogicSetName*]{*Result*(*Operator*) *ParameterList*} *Result:* is T or F, T is True.
Cmpr Excp	Thrown an Exception when match a compare.	WITLOG(ThenFailed) [*LogicSetName*] {*ExceptionMessage*}
CompareEnd	End of a compare	WITLOG(CompareEnd) [*LogicSetName*] {========(*Result*)====} *Result:* is TRUE or FALSE.
CchCmprPut	Put a compare result into Cache	WITLOG(CchCmprPut) [*LogicSetName*] {*CompareID*:*result*}
CchCmprGOT	Got a compare result form Cache	WITLOG(CchCmprGOT) [*LogicSetName*] {*CompareID*:*result*}

10. Compare Fail Log

Normally, Compare Log above explained only log out the compares succeeded. If open this Compare Fail Log, fail compares will be log out too.

11. Compute Log

Log of variables compute trace and result.

Information type	Meaning	Format(in one line)
ComputeRlt	Result of one compute	WITLOG(ComputeRlt) [*LogicSetName*] {*VariableName*=*Value*}
SkippedExp	An expression compute is skipped. Normally, the reason is part of this expression got exception.	WITLOG(SkippedExp) [*LogicSetName*]{*Reason*}
CachVarPut	Put a variable compute result into Cache	WITLOG(CachVarPut) [*LogicSetName*] {*VariableName*:*Value*}
CachVarGot	Got a variable compute result from Cache	WITLOG(CachVarGot) [*LogicSetName*] {*VariableName*:*Value*}

Open/Close Log

Using Null logger

For the best performance of a system which needs not log at runtime, Witmate engine offers a null-logger.jar. This logger implements all logc4j interfaces which used by Witmate engine with do nothing code. Just uses null-logger jar to replace logc4j.jar or log4j2me.jar that is used by Witmate engine to switch to null logger.

Log4J approach

Witmate engine uses Log4J2ME and Log4J to log out Logs, and Witmate engine Logs are Debug level. So setting Log4J log level to level higher than Debug is an approach to close Witmate engine log.

Control by type

As explained above, these openXXX/closeXXX methods in Witmate.core.kernel.Logger can open/close Logs by log type. For example: Logger.openProcLog(); will open process logs.

Control by LogicSet

For complex logics, open or close the Logs of a certain LogicSet is useful. The methods openXXX/closeXXX with LogicSet parameter can use to control Logs by LogicSet.

For example:

```
# Close Process log of lsA LogicSet, open other LogicSets.
Logger.openProcLog();
Logger.closeProcLog("lsA");

# Open Process log of lsA LogicSet, close other LogicSets.
Logger.closeProcLog();
Logger.openProcLog("lsA");
```

Simple Logic language (SL)

As one of logic describing format, Simple Logic language is a human friendly format fitting to create/modify logics by fields' experts and programmers.

First look

SL is designed to be understood / used easily by anyone. The well-known IF-THEN pattern is used to describe logics in SL, and it includes expressions that make it easy to express logic with application experts' vocabularies.

This is a simple logic set that will return different result according to content of input_text. input_text, input_1 and input_2 are inputs when run this logic set. Comments of SL are any text start from '#' to end of that line.

```
LogicSet FirstLS
   # Define variables type with Def command.
   Def int input_1, input_2, sum
   Def text input_text

   Let sum=input_1+input_2
   If input_text="get sum" Then Return sum
   If input_text="check" And input_1>input_2 Then Return "Bigger"
```

It is clear that the SL is very easy to understand, logics are described clearly even without any comments about them. Below is the example code to run this logic set:

```
String RULE_SERVICE_PROVIDER = "Witmate engine ";
Class provider=Class.forName( "Witmate engine .core.kernel.ServiceProvider" );
//Get the rule service provider from the provider manager.
RuleServiceProviderManager.registerRuleServiceProvider(RULE_SERVICE_PROVIDER,provider
);
serviceProvider=
RuleServiceProviderManager.getRuleServiceProvider(RULE_SERVICE_PROVIDER);
//get the RuleAdministrator
ruleAdministrator = serviceProvider.getRuleAdministrator();
RuleExecutionSet
   res=ruleAdministrator.getLocalRuleExecutionSetProvider(null).createRuleExecutionSet(
   in_stream, prop );
in_stream.close();
//Register the RuleExecutionSet
String uri = res.getName();
ruleAdministrator.registerRuleExecutionSet(uri, res, null );
//Get a RuleRuntime and invoke the rule engine.
RuleRuntime ruleRuntime = erviceProvider.getRuleRuntime();
//create a StatelessRuleSession
StatelessRuleSession
   statelessRuleSession = (StatelessRuleSession) ruleRuntime.createRuleSession(uri, new
   Hashtable(), RuleRuntime.STATELESS_SESSION_TYPE);
//Execute the rules without a filter.
Vector results = statelessRuleSession.executeRules(null);
//Release the session.
statelessRuleSession.release();
```

The in_stream should be an input stream from network connection, file, string or database.

About details of WitME API, refers the chapter of WitME API.

Important concepts

As a logic language, there are some concepts of SL that need to be understood before starting to use it.

Non-process language

SL is NOT a process language, that means engine does NOT execute code following the order by that the code are written. Not like Java, C++ or any process languages, SL just describe logics, but the order of execution of this logics are decided by Witmate engine.

SL offers some language elements to control order of execution, as an extension of logic language. Special for Then/Else blocks, commands in blocks will be executed following a certain order. Refer If/Then command about details.

Variable assignment

Variables Scope

There are 5 scopes for variables in SL. Scopes have priority that variables with higher priority can refer to variables with lower priority. To variables which have same name, the value of variables is decided by the variable with highest priority. Below is the scope list by the priority from high to low:

1. Input variables: Input at runtime
2. Block variables: Assigned in Then/Else block.
3. LS variables: Assigned in Logic Set out of Then/Else block
4. Referred variables: Assigned in Logic Set named LS1 out of Then/Else block that is referred by another Logic named LS2. When firing LS2, variables that assigned in LS1 are in referred variables scope.
5. Public variables: Published by Publish command.

Variable Computation

- Variables of scopes besides of blocks are computed when they used by logic

As the “sum” variable, it defined by “Let sum=input_1+input_2”, but only when the input_text=”get sum” logic is true, “sum” will be computed.

- Variables of block scope are computed according to their sort number order.

Refer Let command for details.

Basic elements

Witmate SL is free context language to describe logics. These are basic elements of it.

Commands

Basic functional part of SL is command. Words of commands and their parameters are reserved by SL and can NOT be used as names in SL, they are: LogicSet, Cut, Def, Let, If, Then, Else, End, Publish, Do, Fire, Return, ReturnC, Refer, ResultCount, CacheLevel,

VarCacheSize, CompareCacheSize. All Commands are insensible. So all of CUT, cut, CUt, cUt or more are parsed as Cut command.

Grammar and usage of commands will be introduced in segments below.

Comments

SL has two types of comments:

- To EOL comments are any text between character '#' and end of line. These are right examples:

```
# Start a logic set
# Define variables type with Def command.

Def int input_1, input_2, sum #Define integer variables.

Let sum=input_1+input_2 # Assignment
If input_text="get sum" #get sum?
Then #Get sum.
Return sum
```

This example will get compile error, because comment out the "Then" command

```
If input_text="get sum" #get sum? Then
Return sum
```

- Block comments are any text included end of line between two '##' as:

```
Def int V1 ## This is a
Block comments.##
Let V1=100+##15##9 ##The value of V1 will be 109##
```

Names

Names of SL must follow these limits:

- Characters are '_', '$', '.', '0-9', 'a-z' or 'A-Z'
- Can NOT start by '0-9', '$', or '.'

Semicolon

Semicolon is the separator between commands in one line. For example:

```
Let A=1+3; Let B=A+C; If B>12 Then Return 1; If B>30 Then Return 2
```

This format can be used to write simple logics more compactable and more commands on one page to improve the readability of SL.

Constants

Constants format:

Format	Type	Example	Comments
Numbers	Integer or Long	100 -123456 +9876	
Numbers with dot	Float or Double	100.123 -0.5678 +100.5698	
Any text around by double quotation marks	Text	"A text" "A text with ¥¥¥" "	
Date/time format around single quotation marks	Date		Refer below.
True or False	Boolean	true faLse	Insensible

Support date formats:

- yyyy/MM/DD hh:mm:ss:SSS, yyyy/MM/DD hh:mm:ss:SSS *TimeZone,* yyyy/MM/DD hh:mm:ss, yyyy/MM/DD hh:mm:ss *TimeZone*,
- yyyy/MM/DD hh:mm, yyyy/MM/DD hh:mm *TimeZone*, yyyy/MM/DD, yyyy/MM
- yyyy-MM-DD hh:mm:ss:SSS, yyyy-MM-DD hh:mm:ss:SSS *TimeZone*, yyyy-MM-DD hh:mm:ss, yyyy-MM-DD hh:mm:ss *TimeZone*,
- yyyy-MM-DD hh:mm, yyyy-MM-DD hh:mm *TimeZone*, yyyy-MM-DD, yyyy-MM
- yyyyMMDDhhmmss, yyyyMMDDhhmmss*TimeZone*

Month and day lacked will be set to 1. Minute, second and millisecond lacked will be set to 0. *TimeZone* is Time zone ID as GMT, PST or JST etc.

Example: 2005/1/1 12:33:00 PST, 2005-1-2 10:10, 20050101123300PST

Variables

Variables of SL must be defined by Def command, and follow the naming limits. Names of Variables can NOT same as commands and true/false. Refer the "Variables assignment" segment in "Important Concepts" chapter and Def command for details.

Operators

Witmate engine offers many operators for calculate and compare. And it's possible to create new operators by users. Besides of basic operators, other operators should be used as

function format. Refer Expression segment below about basic operators definition.

Witmate engine offers operators in 3 packages:

- witmate.engine.opt in witcore.jar or witcore-all.jar: Normal operators
- witmate.engine.opt.ex in witopt-ex.jar: extended operators
- witmate.engine.opt in witpeel.jar: Normal operators on J2SE 1.4.1_02 or higher platform

Details about operators refer Java Doc of these packages above.

Expression

There are 2 types of expression in Witmate engine: Compute expression and Compare expression. Compute expression is used in variable assignment (Let command) to calculate a variable. Compare expression is used in logic (If command) to check logics.

	Compute Expression	Compare Expression
Basic operators	+, -, *, /	<, >, =, <=, >=, <>, And, Or, Not
Functions or Customized operators	Usable	Usable
Type of Return	All data types of SL	Boolean

Functions / operators

Witmate engine treats functions as same as operators. To create a function is same as new an operator. In this manual, functions, operators and customized operators figure out same thing, besides of the special explain about basic operators.

About how to create a function or customized operator refer “How to define customized operators/functions” topic in How to chapter.

Grammar

SL grammar is defined with Witmate engine BNF. The syntax of Witmate engine BNF is:

- {}: 0 to many; [] 1 to many; <>: 0 or 1; (): Must and only 1.
- All elements need more definition are start with _
- End items well known are included in ' '.
- Key words are words besides of 2 types of items above. All key words are insensible.

The BNF is:

```
SL ::= [ _LogicSet ]
_LogicSet ::= LogicSet _LogicSetName < Refer _ReferLogicSetNameList > < ResultCount ( '-1' | ['0-9'] ) >
< _CacheParameters > < _TypeDef > < _CutNames >    _LogicDefinition
_CacheParameter ::= < CacheLevel ( '-1' | ['0-9'] ) > < VarCacheSize ( '-1' | ['0-9'] ) > < CompareCacheSize
( '-1' | ['0-9'] ) >
_TypeDef ::= Def _Type _VarFunName {, _VarFunName}
_VarFunName ::= _VarName | _FunctionName
_Type ::= _BasicType | 'java class name'
_BasicType ::= Int | Integer | Long | Float | Double | Real | Text | String | Str | Date | Bool | Boolean
| Object | Obj
_CutNames ::= Cut _FunctionName _FunctionName
_LogicDefinition ::= [ _VarAssignment | _RuleDefinition ] < Else _CommandsBlock < End > >
_ReferLogicSetNameList ::= _LogicSetName { ,_LogicSetName }
_VarAssignment ::= < _LetSort > Let _VarName = ( _VarName | _Value | _Expression )
_LetSort ::= ':' _SortNumber ':'
_Expression::= _parameter { _BasicOperator _parameter }
_BasicOperator::= + | - | * | /
_parameter ::= _Value | _VarName | _Function
_Function ::= _FunctionName '(' < _FunParamList > ')'
_FunParamList ::= ( _Value | _VarName | _Function ) { , < _Value | _VarName | _Fucntion > }
_RuleDefinition ::= < _RulePriority ><'{' _RuleName '}'> If _LogicFormula Then _CommandsBlock < End
>
_RuleName ::= [ 'a-z' | 'A-Z' | '_' ] { 'a-z' | 'A-Z' | '0-9' | '_' | '.' | '$' }
_RulePriority ::= ':' SortNumber ':'
_LogicFormula ::= _LogicItem { ( AND | OR ) _LogicItem }
_LogicItem ::= NOT '(' _LogicFormula ')' | _LogicFormula | NOT '(' _LogicAtom ')' | _LogicAtom
_LogicAtom ::= _parameter { _BasicCompare _parameter }
_BasicCompare ::= = | > | < | >= | <= | <>
_CommandsBlock::= { _VarAssignment }   < _PublishCommand > < _DoCommand > ( _FireCommand |
_ReturnCommand )
_PublishCommand ::= Publish _VarName {, _VarName}
_DoCommand ::= < _DoOrder > Do _Expression
_DoOrder ::= ':' _SortNumber ':'
_FireCommand ::= Fire _LogicSetName '(' < _FireParamList > ')'
_FireParamList ::= ( _VarName ) { , _VarName }
_ReturnCommand ::= ( Return | ReturnC ) < _ReturnParamList >
_ReturnParamList ::= ( _Value | _VarName ) { , ( _Value | _VarName ) }
_LogicSetName ::= [ 'a-z' | 'A-Z' | '_' ] { 'a-z' | 'A-Z' | '0-9' | '_' | '.' | '$' }
_VarName ::= [ 'a-z' | 'A-Z' | '_' ] { 'a-z' | 'A-Z' | '0-9' | '_' | '.' | '$' }
_Value ::= ( _Number | _Text | _Date | _Boolean )
_Number ::= < + | - > ['0-9'] < .['0-9'] >
_IntegerNumber ::= < + | - > ['0-9']
_SortNumber ::= ['0-9']
_Text ::= "" { 'Any Characters' } ""
_Date ::= '" DateText '"
_Boolean ::= ( true | false )
_FunctionName ::= [ 'a-z' | 'A-Z' | '_' ] { 'a-z' | 'A-Z' | '0-9' | '_' | '.' | '$' }
_EOL ::= ';' | 'End of Line' | 'End of File'
_Commnets ::= #{ 'any text' }( 'End of Line' | 'End of File' )
_BlockComments ::= ##{ 'any text' | 'End of Line' }##
```

Commands

This segment explains all commands of SL one by one.

All of Space, Tab and End of Line are usable separator of items of commands. So all of below code are OK:

```
If input_text=”get sum” Then Return sum

If input_text=”get sum”
Then Return sum

If       input_text=”get sum”
Then  Return sum

If
input_text
=
”get sum”
Then
Return
sum
```

LogicSet

Syntax:

LogicSet *LogicSetName*

Refer *ReferLogicSetList*

ResultCount *IntegerNumber*

CacheLevel *IntegerNumber*

VarCacheSize *IntegerNumber*

CompareCacheSize *IntegerNumber*

LogicSet is the command to start a logic set. All logic set must start with this command. Every LogicSet must have a unique name in one execution session. If load two logic set with same name, second one will replace the first.

Other parameters of LogicSet command are optional.

- Refer parameter figure out refer logic set by this one. *ReferLogicSetList* is logic sets names separated by comma. When run this logic set, logics of these refer logic sets, If commands, will be run at first, and variables assignments, Let commands, will be refer if can not find assignment for a certain variable in this logic set. The feature of refer logic sets is usable for some general logics and variable assignments referred by more than one logic set. Refer logic set itself is not error, but it is not a normal usage of Refer parameter.
- ResultCount parameter tells Witmate engine engine stop run this logic set after gotten how many results. If not this parameter, as default, Witmate engine will go through all logics of this logic set and get all possible results. Normal usage of this parameter is set to 1 to tell Witmate engine get only one result from this logic set. Zero is a legal value, but generally it is only for debug or test for a complex firing among logic sets. Set ResultCount to 0 of one of these logic sets to test whether can get certain results. Any integer is OK, so ranking results, as getting top 10 best results, are possible.

- CacheLevel parameter controls open/close cache for this logic set. 0 closes and any non zero value opens cache for this logics set. Cache does not help to get high performance for all types of logic sets. Going through a simple logic set logics may be take shorter time to operate cache. Small devices may have not enough fast memory to cache. Default value is 8, and this default value can be configured by changing configurations of Witmate engine, refer Configuration chapter for details.
- VarCacheSize parameter set the cache size of variable. Witmate engine has to types of cache: variable cache and compare cache. Variable cache caches the result of variable assignments to save time to compute a same variable many times. Compare cache caches the result of logic compares to save time to re-compare a same compare. To use memory well for different environments and logics, it is required to control these cache sizes. Both of the size of these caches can be controlled in SL. This parameter is the number of variable will be cached, if over this number the variable less used will be removed from cache. Default is unlimited.
- CompareCacheSize parameter controls the size of compare cache. As explained above, his parameter can be used to use well limited memory in small devices. Default is unlimited.

Examples:

A very simple example:

```
LogicSet LS_A
```

A full example:

```
LogicSet LS_big
        Refer LS_A, LS_B, LS_C ResultCount 3
        CacheLevel 1 VarCacheSize 30 CompareCacheSize 50
```

Def

Syntax:

Def *Type NameList*

Def is from “define” to define type of variables and functions.

Type is one of basic types or a java class full name.

Basic types are:

Type name	**Alias**	**Java Type**
Integer	Int	java.lang.Integer
Long		java.lang.Long
Float		java.lang.Float
Double	Real	java.lang.Double
Text	String, Str	java.lang.String

Date		java.util.Date
Bool	Boolean	java.lang.Boolean
Object	Obj	java.lang.Object

Type of class full name is for input variables that have not name, so to look for a certain variable, must according to its class. This is the approach of JSR-94 to identify input values. Witmate engine offers another more clear and powerful approach to identify input values that is the input variable approach. The input values are set into an instance of Witmate engine .core.kernel.elements.ConstantVariable with a unique name among this input values. Logics look for certain value input by the name of variables. So input more than one same class inputs is possible, for example, this situation is very normal that input customer purchase total amount and tax rate into a logic set.

All variables is used in logic set must be defined with type by Def commands. Functions type Def is optional. If SL compiler can access the class of certain function, compiler will get function type by itself; otherwise Def command is needed for this function. This is feature gives system developers a tool to separate runtime system and logic create/management system by only deploying functions definition class/JAR files into runtime system and define type of functions by Def command in SL.

Examples:

```
Def integer int1, int2
Def str strVar1
Def com.my.element.class1 cl1
Def text function1
```

Let

Syntax:

: *SortNumber* : Let *VariableName* = *ComputeExpression*

Assign variables; figure out how to compute a variable.

SortNumber is optional. It has different meanings in and out Then/Else block. In Then/Else block, it is the execution order of Let commands. All Let commands without sort number will be executed at first, then execute Let by order of *SortNumber* from small to large. But out of blocks, it is not the execution order of Let command; it is a reference control sort. The rule is that variables in smaller sort Let commands can refer the Let with larger sort number, but larger can not refer smaller. Check example below, assuming the public variable pub_int1 is 8. (Details about pubic variable refer Publish command)

```
LogicSet ls_let_test
Def int pub_int1, sum
:10: Let sum=pub_int1+10
:20: Let pub_int1=0
If sum>11 Then Publish pub_int1; Return sum
```

This logic set will return a result of integer 18. But if there is not sort numbers of Let commands, the sum will never larger than 11, because when calculate sum, Witmate engine will refer "pub_int1=0". By firing another logic set, this problem can solute without Let sort number feature. But this feature makes it easier to describe some normally used pattern. Default sort number is 0. So all Let commands without sort number will be calculated at first. Order of calculations with same sort number is not defined, by another words order may be changed time by time. Serial sort number is not required.

Here is an extreme example below: assuming the input variable in_int1 is 28

```
LogicSet ls_let_test1
Def int in_int1
Let in_int1=6
If in_int1<10 Then Let in_int1=in_int1+10; Return in_int1
Else Return 0
```

The result is 38. Any expression which includes the variable with the same name of assigned variable, as x=x+1, figures out the variable x in expression is an input variable or public variable. So "in_int1<10" is true, and "Let in_int1=in_int1+10" in THEN block has higher priority to load in_int1 in expression from input.

Variable name must be a legal name of SL, and its type must be defined by Def command.

Examples:

Let int1=12

:1:Let int2=12 + int1 * 10 / int1

:16:Let text1= Witmate engine .opt.Text$ToLowerCase (“OK”) + “!”

If Then

Syntax:

: *SortNumber* : { *LogicName* } If *CompareExpression* Then *CommandsBlock* End

Describe logics in SL. It is the most important command of SL.

SortNumber is optional. Witmate engine will check logics from smaller sort number to bigger. Default value is 0. All logics without sort number will be checked at first. Order of execution of logics with same sort number is not defined, by another words order may be changed time by time. Serial sort number is not required

LogicName is optional. This is the name of logic; it may be same in one Logic Set. It must follow the naming limits of SL. Name of logic is usable for:

- More readable SL: It is easier to figure out certain logic for logic management and modifications by name, special for the case to communicate with peoples.
- Linking certain logic between SL and SLML: Because logic name is compiled into SLML, it is clear which SL format is compiled into which SLML format by logic name.
- Identify logic in SLML: Identify certain logic in SLML for logic exchange or 3rd party extensions by logic name.
- Trace the result of certain logic in Log: Logic name is outputted in Log, so it is useful to trace the result of certain logic.

The order of *SortNumber* and *LogicName* appeared in SL is free, so both of examples below are OK.

:12:{LogicA}If 1>v1 Then return 1

{LogicB}:2:If 2<v2 Then return 2

CompareExpression is the expression to describe the logics using compare operators and functions.

CommandsBlock are commands that will execute when the *CompareExpression* is true. It may include Let command, Publish command, Do command, and Return or Fire command. One *CommandsBlock* only can include one Return or Fire command, and one Publish command. The Let commands in *CommandsBlock* will over load the Let out of the If Then command, if they assign a same name variable.

Execution order of commands in *CommandsBlock* is:

1. Let commands
2. Do commands

3. Publish command
4. Return or Fire command

End is optional. For most cases it does not required. But check the example below:

```
If int1>10
Then
  Return "1"
End
Let int2=12
If …
```

If there is not the "End", it is not clear that the "Let int2=12" is not a part of If Then command.

Examples:

If 1>2 then Return "Never"

:12: If int1>12

Then

Let int2=6*int1; Publish int2

Fire LS_x(int1)

: 6 :{Always}If 2>1 then Return "Always"

Publish

Syntax:

Publish *VariableNameList*

This command only can be used in *CommandsBlock* of If Then command. And can only be used one time in one *CommandsBlock*. There is one public area in which variables are public among logic sets in one session. Publish command publish variables in *VariableNameList* into this public area. Publish a same name variable will replace the previous one.

VariableNameList is variable names separated by comma.

Public variables are useful to pass variables cross logic sets.

Examples:

Publish pub_int1

Publish int1, text2, pub_int1

Do

Syntax:

: *SortNumber* : Do *ComputeExpression*

This command only can be used in *CommandsBlock* of If Then command. Calculate a *ComputeExpression*. Because *ComputeExpression* may include any customized

operator/functions, Do command can be used to call outside processes. Refer "How to call a outside process" segment for details.

SortNumber is optional. Witmate engine will execute do commands from smaller sort number to bigger. Default value is 0. So all Do without sort number will be executed at first. Order of execution of Do commands with same sort number is not defined, by another words order may be changed time by time. Serial sort number is not required.

Examples:

Do 1+2

:1:Do Stack$Push(in_stack, "1")

: 19 :Do my.process.WriteDB("the text written")

Return/ReturnC

Syntax:

Return *VarValueList*

Or

ReturnC *VarValueList*

This command only can be used in *CommandsBlock* of If Then command. Can NOT be used together with Fire command in one *CommandsBlock* . And can only be used one time in one *CommandsBlock*.

Return results list.

VarValueList is optional, even an empty results list is not normal. It is values and variables separated by comma. All variables must be defined by Def command.

Return command returns the value object instances of values or variables. This is a JSR-94 type return that uses Class and order of result instances to identify the results,

ReturnC command returns the instances of witmate.engine.core.kernel.elements.Constant for values and witmate.core.kernel.elements.ConstantVariable for variables. As extended by Witmate engine, this return type makes identify results easier by code, because the Witmate engine .core.kernel.elements.Constant has a value type number, and the Witmate engine .core.kernel.elements.ConstantVariable has the variable name.

Examples:

Return

Return 1, "a text", 4.5, '2005/10/07'

Return var1, 12, var2

RetunC

ReturnC "a text", 1

ReturnC var1, 18, var2

Fire

Syntax:

Fire *LogicSetName* (*VariableList*)

This command only can be used in *CommandsBlock* of If Then command. Can NOT be used together with Return/ReturnC command in one *CommandsBlock* . And can only be used one time in one *CommandsBlock*.

Fire a logic set. Fire the same logic set that boots the Fire command is legal.

LogicSetName is required, and must be loaded into session previously at runtime. It must follow the naming rule of Witmate engine.

VariableList is variables separated by comma. These variables must be defined by Def command, and will be the inputs of the logic set fired. It is optional.

Examples:

```
Fire logic_set1( )
Fire ls_2( var1, var2 )
```

Else

Syntax:

Else *CommandsBlock* End

Set default *CommandsBlock* for one logic set. One logic set can only use one Else command. Else command tells the Witmate engine: if all rules are failed, run these commands.

It is optional for logic set.

Usable Commands and process algorithm of *CommandsBlock* are totaling same as *CommandsBlock* in If Then command.

End is optional, and has same usage as End in If Then command.

Examples:

```
Else Let va=1; Retrun va
Else
    Do myProcess()
    Fire ls1()
End
```

Cut

Syntax:

Cut *originalFuntionName newFunctionName*

Rename a function name to a new one. The name of "Cut" is come from the original usage that cut a long java class name to a shorter name.

Both of *originalFuntionName* and *newFunctionName* required.

Cut commands can be used at 2 situations:

- Shorter a long full java class name for operators/functions. To get clear, more readable logic expression. For example:
  Cut com.my.extend.operator.GetCustomerName GetCName
- Naming experts' vocabulary. Details about experts' vocabularies, refer "How to define experts' vocabularies" segment.
  Cut com.my.extend.operator.WriteCustomerDB saveCustomerProfiles

Examples:

Cut com.my.extend.operator.GetCustomerName GetCName
Cut com.my.extend.operator.WriteCustomerDB saveCustomerProfiles

Logic set Examples

This is the logic set for Stateful session test case of JSR-94 TCK:

```
LogicSet RuleExecutionSet2
   Def org.jcp.jsr94.tck.model.Customer Customer
   Def org.jcp.jsr94.tck.model.Invoice Invoice
   Def int credit_limit, amount
   Def str invoice_status
   Cut Witmate engine .jsr94tck.opt.SetCustomerCreditLimit setCreditLmt

   Let credit_limit=GetAttr(Customer,"CreditLimit")
   Let amount=GetAttr(Invoice,"Amount")
   Let invoice_status=GetAttr(Invoice,"Status")

   If credit_limit>amount and invoice_status="unpaid"
   Then
         :1:Do setCreditLmt(Customer,credit_limit-amount)
         :2:Do SetAttr(Invoice,"Status","paid")
         Return Customer,Invoice

   Else
         Return "Input may be wrong!"
```

The GetAttr function is the short name of Witmate.opt.GetAttribute operator of Witmate.opt package in WitPeel.jar. Details about operators refer Java Doc of this package.

Simple Logic Markup Language (SLML)

SLML is another format of logics for computers processes or communications between hosts. The explanations about SL commands are correct for SLML tags too. The best way to learn SLML is write some SL logic sets and compile them to SLML.

SLML Format

First Look

This is a simple SLML:

```
<?xml version="1.0"?>
<slml>
    <lset id="ASet">
        <let n="Y" t="int">
            <atom opt="Witmate engine .opt.Math$AddInt">
                <con t="int">1</con>
                <var t="int" n="X" />
            </atom>
        </let>

        <lgc>
            <if>
            <and>
                <atom opt="Witmate engine .opt.Logic$True" />
            </and>
            </if>

            <then>
                <let t="int" n="X">15</let>
                <rtn>
                    <var n="Y" t="int" />
                </rtn>
            </then>
        </lgc>
    </lset>
</slml>
```

Schema of SLML

Refer slml.xsd file.

SLML must begin with <?xml …, if the Wit-SL-InType property is not set, when calling engine API. Details about Wit-SL-InType property refer WitME API segment.

SLML Parser

SLML parser load logics from SLML format into Witmate engine. At same time, SLML parser optimizes the logics for runtime performance.

Optimize logic set

SLML parser optimizes logic set by these approaches:

- Ignore constant comparisons in complex comparison

- Integrate same comparisons leveraging isSeriary method and isDynamic method of operators

WitDebugger

WitDebugger is a text console based debugger of Witmate engine . As an extension to jdb the Java debugger shipping with JDK, It has these advantaged features:

- Support both of local JVM and remote JVM debug. This makes the debug on the Witmate engine running on mobile/small devices possible.
- Seamless integration with Java code debug offers the seamless tracing from Witmate engine code into customized operations code.
- Small footprint makes it to be a usable tool in real world environment debug, the resource mostly are limited for debuggers in real world environments.

Usages

WitDebugger is extended from jdb, so most of usages of WitDebugger are same as jdb.

Example start command of WitDebugger to debug Witmate engine on local JVM

```
witdb.bat -sourcepath your¥Witmate engine ¥SL¥src -sourcepath -classpath your¥WitSDK¥lib¥witcore-all.jar;and¥other¥Witmate engine ¥jars¥your¥app¥needs your.main.class.to.call.Witmate engine
```

Example start command of WitDebugger to debug Witmate engine on remote JVM

```
witdb.bat -sourcepath your¥Witmate engine ¥SL¥src -connect com.sun.jdi.SocketAttach:hostname=localhost,port=5000
```

For more details, refer the jdb document in your JDK.

Commands

All commands which are offered by jdb are usable in WitDebugger. And WitDebugger extends these commands to support Witmate engine debug.

wstop

List and set breakpoints at commands besides of the Let command. It has several usages as below:

- wstop

  List all Witmate breakpoints.
- wstop at <logic set name>:<command id>[@<uri>]

  Stop at a command. Uri is optional. WitDebugger will stop at all commands that are figured out, without checking the uri that LogicSet is loaded, if the uri isn't set.

  The command id normally is line number in SL or the cid attribute in SLML.

```
wstop at test_logicset1:12
wstop at test_logicset1:12@the.special.uri
```

- wstop inthen <logic set name>[@<uri>]

  Stop before any then/else block is fired in the logic set figured out.

```
wstop inthen test_logicset1
wstop inthen test_logicset1@the.special.uri
```

- wstop inset <logic set name>[@<uri>]

  Stop when the logic set is fired.

```
wstop inset test_logicset1
wstop inset test_logicset1@the.special.uri
```

wclear

List/Remove the breakpoints. To different types of breakpoints set by wstop commands, clear command has these usages:

- wclear

  List all Witmate engine breakpoints.
- wclear at <logic set name>:<command id>[@<uri>]

  Remove the stop at break point.
- wclear inthen <logic set name>[@<uri>]

  Remove the stop inthen breakpoint.
- wclear inset <logic set name>[@<uri>]

  Remove the stop inset breakpoint.

wlocals

List all variables which can be accessed, when stopped at a Witmate breakpoint.

wlist

List SL source codes. At the first time to run this command, it takes some time, according to the total size of your SL source codes, to parse source codes. The folders of SL code are set by the parameter '-sourcepath' of WitDebugger or set by 'use' command. The SL files must has extension name '.sl'.

wwhere

Dump the current firing tracing. At the first time to run this command, it takes some time, according to the total size of your SL source codes, to parse source codes.This command is usable, even the current breakpoint is out of Witmate engine logic sets. This offers the way to know how Witmate engine goes into your code mostly which is customized operations you wrote.

wstep

Step to next Witmate engine command.

Typical using scenario

There is a typical scenario to use WitDebugger

1. Boot WitDebugger

```
witdb.bat -sourcepath your¥Witmate engine ¥SL¥src -sourcepath -classpath your¥WitSDK¥lib¥witcore-all.jar;and¥other¥Witmate engine ¥jars¥your¥app¥needs your.main.class.to.call.Witmate engine
```

2. Set breakpoints

```
wstop at test_logicset1:12
wstop inthen test_logicset2@the.special.uri
```

3. Run Witmate engine, and stop at a breakpoint.

```
run
WitBreakpoint hit: …
```

4. Using wlocals, wwhere and wlist commands to debug.
5. Using wstep, wstop and 'cont' to run into another location
6. Run to end of your application by 'cont'
7. Exit WitDebugger by 'exit' command.

How to

This chapter explains the solutions for special situations.

System

How to open/close cache

There are 2 ways to open/close cache.

- Set in logics definition.

It is possible to control cache in SL and SLML.

CacheLevel option of LogicSet command in SL controls and clvl attribute of lset tag in SLML control cache status.1 will open cache and 0 will close.

SL example:

```
# Open cache
LogicSet LS01 CacheLevel 1 …

# Close cache
LogicSet LS02 CacheLevel 0 …
```

SLML example:

```
<!-- Open cache -->
<lset n=”LS01” clvl=”1”…

<!-- Close cache -->
<lset n=”LS01” clvl=”0”…
```

- Set null cache implementation

There is a null cache implementation in Core Engine. The null cache just is an empty implementation of cache interface. Set witmate.core.LevelCache configuration key of Witmate engine configurations to witmate.core.kernel.interfaces.NullLevelCache will close cache and to witmate.core.kernel.cache.OneLevelCache will open cache. Refering Configuration segment to know more about Witmate engine configuration.

Because all methods of null cache are just a return without any process, it is faster than the way to set in logics definition. And null cache is included in Core Engine, so you need not to buy the license of cache.

How to define customized operators/functions

Both of operators and functions are same in Witmate engine, and dealt as operators.

Customized operator class needs to implement interface Witmate.opt.Operator.

Key method is “go”, and all process should be coded in this method.

The method isSeriary tells Witmate engine whether your operator is seriary. This method will be called by SLML parser to optimize logics. Changing the parameters order of Seriary operators will change the result of operator, for example "=" is not seriary, should return false in isSeriary method, and ">" is serairy, should return true.

The method isDynamic tells Witmate engine whether the result of your operator is changed even input is same.

Points that pay attention to define customized operators:

1. The “go” method must be thread-safe.
2. The class of operator must can be generated through Class.newInstance() method.

How to call a outside process

Outside process means some application process out of Witmate engine, as sending a e-mail, access database, etc. There are several ways to call outside processes

Using do command

1. Create a customized operator whose go method do the outside process.
2. Add do command with this operator and its operands
3. Using sort number of do command to control sort of execution.

* Do commands will be executed before publish commands.

Using variable computation

1. Create a customized operator whose go method do the outside process.
2. Write a Let command whose expression uses this operator.
3. Add the variable name of the Let into publish, fire or return parameters list in then block, or compare expression for logic compares.
4. If there are more then one processes, write one Let for each process.
5. The order of variable name of Let in fire or return parameters list is the process order.

* Only these operators whose variable name of Let is appear in parameters list of fire, return or compare expressions will be called.

Pay attention:

1. isDynamic of customized operators should be true to enforce Witmate engine not to cache the result of process.
2. isSeriary of customized operators should be true to make the mistake of wrong order of parameters can be found easier

How to define experts' vocabularies

There are 2 approaches to define vocabularies in SL:

- Variables naming

Define variables names by experts' vocabularies rather than IT words by Def command. For example, define a variable as "monthly_cost_sum", not "cost_sum_filed_data".

And it is possible to bridge IT world and experts' world using Let command, as:

```
Def int cost_sum_filed_data, monthly_cost_sum
Let monthly_cost_sum= cost_sum_filed_data

If monthly_cost_sum>…
```

- Operator naming

Naming a method to an experts' vocabulary by Cut command, as:

```
Cut com.my.lib.opt.ReadCostData GetCost

If GetCost(…
```

How to use SLML validation

Although Witmate engine SLML parser checks SLML syntax and log out all mistakes, SLML is defined by a XML schema at http://www.witmate.com/slml, so it's possible to validate SLML with XML validating. For time costing of validation, the default status of validation is closed. The steps to open validation:

1. Confirm your XML parser supports validation. If it does not, Witmate engine will skip the validation.
2. There are 2 approaches for developer and administrator:
   - Set Wit-SL-InType property to "4" when call createRuleExecutionSet/ createRuleExecutionSets method to create rule set(s) in your code. This step gives programmers to control validation when developing.
   - Set VALIDATE_SLML configuration key to 1. The key of this configuration in witmate.properties is "witmate.core.SLMLValidating". This step gives system administrators the possibility to open/close validation after system deployment.

End user tools

How to use sl2slml and slml2ws tools

Sl2slml and slml2ws are two SL/SLML/WitStream translating open source tools.

Use this command to translate SL to SLML:

```
java -cp sl2slml.xxx.jar;wittools.xxx.jar;witcore-all.xxx.jar;slcompiler.xxx.jar;log4j-1.2.9.jar
witmate.tools.SL2SLML <input SL file name> <output SLML file name>
```

And this command to translate SLML to WitStream:

```
java –cp slml2ws.xxx.jar;slmlparser.xxx.jar;wittools.xxx.jar;witcore-all.xxx.jar;log4j-1.2.9.jar
witmate.tools.SLML2WS <input SLML file name> <output Wit Stream file name>
```

xxx is the version number of Witmate engine to use.

Performance tuning

How to get better logic optimization

Follow these guild lines below to gain a better logic optimization result by SLML parser.

- Use same expressions for same logics. For example, use “varA>12” any where in a logic set rather than use both of “varA>12” and “12<varA”.
- Define non seriary operators as possible as you can.

How to save the log4j execution time

In most deployed system, the log isn’t needed. Even closed log by log4j settings, the log4j code still do some checks. To save this time, Witmate engine offers a null-logger jar that implements log4j interface with do nothing codes. To switch to null logger, just uses null-logger jar to replace logc4j jar or log4j2me jar that is used by Witmate engine.

How to use other languages in logic

Too use other languages, the logic contents must be encoded by UTF-8.