# UBIWARE Platform

## Application Developer's guide

RAB overview

*Artem Katasonov and Michael Cochez*

24.07.2012 version 1.24

Industrial Ontologies Group
University of Jyväskylä
email: michael.s.l.cochez@jyu.fi

# Contents

# 1 Available Reusable Atomic Behaviors (RABs)

This chapter describes RABs available at the moment. More RABs might be added in the future versions of the platform. In newer releases, backward compatibility is preserved as much as possible.

The terminology to describe RABs is as follows:

- Full name: complete name of this behavior.
- Action: Description of the action performed.
- Inputs: which input parameters the RAB expects with the following properties.
    - Name: the resource a value is given for; the name of the parameter.
    - Meaning: the meaning of this parameter.
    - Range: the range of allowed values, the following are used:
        - String : String literal
        - Integer: Integer literal
        - Boolean: possible values are the string literals “true” and “false”.
        - Resource: A valid URI (the implementation might limit to URL)
        - Context container (many RABs allow the general beliefs container of the agent to be specified with the string literal “G”)
        - Filename: the name of a file valid on the underlying file system.
        - Any = a resource, integer literal, string literal or Boolean.
    - Mandatory: Indicates whether this parameter must be specified or is optional.
    - Default: indicates the default value for this parameter (only if not mandatory)
- Implicit inputs: inputs which the RAB reads from the agents believes without the invocation of the RAB explicitly mentioning it.
- Outputs: output printed on the standard output or standard error output.
- Effect: effect of running this RAB on the agent’s beliefs and environment.
- Remarks: notes which should be taken into consideration when using this RAB.
- Examples of usage: examples illustrating use cases for this RAB.
- Libraries: additional libraries needed to execute this RAB. Libraries for execution of RABs delivered with the platform are available in the distribution.

Note that “Outputs” specified do not include stack traces printed by behaviors when exceptions are thrown. If any of the entries (Inputs, Implicit inputs, Outputs, etc…) is not described for an RAB, it indicates that there is no input, no implicit input, no printed output, etc…

In the rest of the text, the following namespace definitions are in use:

```
@prefix p: <http://www.ubiware.jyu.fi/rab_parameters#>.
@prefix java: <http://www.ubiware.jyu.fi/rab#> .
@prefix ex: <http://www.example.com/ubiwareexamples#> .
```

## 1.1 Debugging actions

Two RABs are made available to print output to the console. One RAB opens a debug window in which the internal beliefs' structure of the UBIWARE agent can be browsed. Usage of these RABs is only recommended for debugging purposes.

### 1.1.1 Print

*Full name:* java:ubiware.shared.PrintBehavior

*Action:* Prints to the screen (and to the log) specified text. Text is printed preceded by "[<present time>] <agent's name>: ".

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:print | The text to print. | Any | Yes | |
| p:error | Is this printed text an error? | Boolean | No | "false" |

*Outputs:* "[<present time>] <agent's name> : <the value of p:print>" to standard output if p:error is "false", "<agent's name> reports error: <the value of p:print>" to standard error if p:error is "true".

```
Remarks:
Printing will happen on the console of the container in which the
agent currently resides.
```

*Examples of usage:*

Print "Job is done!" to standard output.

```
{sapl:I sapl:do java:ubiware.shared.PrintBehavior}
sapl:configuredAs
{p:print sapl:is "Job is done!"}
```

Print "An error occurred" to standard error.

```
{sapl:I sapl:do java:ubiware.shared.PrintBehavior }
sapl:configuredAs
{
     p:print sapl:is "An error occurred" .
     p:error sapl:is "true"
}
```

### 1.1.2 PrintBeliefs

*Full Name:* java:ubiware.shared.PrintBeliefsBehavior

*Action:* Prints to the screen (and to the log) the contents of specified context container. The print-out is preceded by "[<present time>] <agent's name>: ".

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:print | The context container, the contents of which are to be printed. If "G" is given, all the beliefs of the agent are printed. | Context container or "G" | No | "G" |
| p:indents | If is equal to "true", the line break and indentation is used to separate beliefs. Otherwise, a single line is produced. | Boolean | No | "false" |

*Outputs:* the beliefs of the specified context container.

*Examples of usage:*

Print all the beliefs of the agent without indentation

```
{sapl:I sapl:do java:ubiware.shared.PrintBeliefsBehavior}
sapl:configuredAs
sapl:null
```

Print certain beliefs of the agent with indentation

```
{sapl:I sapl:do java:ubiware.shared.PrintBeliefsBehavior}
sapl:configuredAs
{
    p:print sapl:is {ex:John ex:hasAge 25} .
    p:indents sapl:is "true"
}
```

*Remark:* In order to print the content of a certain container with indentation, proceed as follows:
Assume we have the following believe in the agent:

```
{ex:John ex:loves ex:Mary .ex:Mary ex:loves ex:Peter}
ex:accordingTo ex:Kate
```

Then the following construct will print the content of the subject container

```
{?variableBindingToTheContainer ex:accordingTo ex:Kate}
=>
{
        {sapl:I sapl:do java:ubiware.shared.PrintBeliefsBehavior}
        sapl:configuredAs
    {
        p:print sapl:is ?variableBindingToTheContainer .
        p:indents sapl:is "true"
    }
}
```

Printing will happen on the console of the container in which the agent currently resides.

### 1.1.3 DebugBehavior

*Full name:* ubiware.core.behaviors.Debugbehavior

*Action:* Opens a debug window into which the agent's beliefs' structure can be browsed and searched. The agent's reasoning loop can also be stopped from this user interface.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:stop | Should the agent stop reasoning immediately upon execution of this RAB? | Boolean | No | "false" |

*Implicit inputs:* All agent's beliefs are taken as an input because all will be visualized.

*Outputs:* None

*Effect:* The debug window can be used for debugging.

*Remarks:* Debugging the agent beliefs might sometimes interfere with the agent reasoning cycle. In that case a ConcurrentModificationException will be printed on the console. Work can be continued normally, though. When the agent dies, this cannot be seen in the debug window. This can only be read from the console.

*Examples of usage:* Open de debugger but do not stop the agent.

```
{sapl:I sapl:do java:ubiware.core.behaviors.DebugBehavior}
sapl:configuredAs sapl:null
```

Open de debugger and do stop the agent.

```
{sapl:I sapl:do java:ubiware.core.behaviors.DebugBehavior}
sapl:configuredAs {p:stop sapl:is "true"}
```

*Libraries needed:* None.

## 1.2 Local actions

This section describes behaviors that do not involve any other agents, i.e. have no social component.

### 1.2.1 ExternalAppStarterBehavior

*Full name:* ubiware.shared.ExternalAppStarterBehavior

*Action:* Starts an external software application. If the command is successfully executed (it does not necessarily implies that the application is started), ends in success. If an exception occurs, ends in failure.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:execute | Command to execute. | String | Yes | |

*Effect:* the RAB will try to execute the command in the environment.

Remarks: This feature is operating system dependant. Furthermore, signals from the executed command cannot be caught with this RAB.

*Examples of usage:*

*Windows:* Execute the text editor Notepad and open in it file "Dog.txt".

```
{sapl:I sapl:do java:ubiware.shared.ExternalAppStarterBehavior}
sapl:configuredAs
{p:execute sapl:is "notepad.exe Dog.txt"}}
```

Linux: Execute the script dog.sh

```
{sapl:I sapl:do java:ubiware.shared.ExternalAppStarterBehavior}
sapl:configuredAs
{p:execute sapl:is "./dog.sh"}}
```

*Libraries needed:* none.

### 1.2.2 UnzipperBehavior

*Full name:* ubiware.shared.UnzipperBehavior

*Action:* Unzips a ZIP archive. If parameter "delete" is equal to "true", deletes the original file. If an I/O exception occurs, ends in failure. Otherwise, ends in success.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:input | Name of ZIP file. | Filename | Yes | |
| p:delete | Whether to delete the ZIP file after unzipping the contents. | Boolean | No | "false" |

*Outputs:* On success, prints to the screen "<input> unzipped".

*Effect:* The archive is unzipped on the file system on which the UBIWARE platform is running.

*Remarks:* Since this feature is accessing the local file system, it is dependent on the underlying .operating system and the container in which the agent resides.

*Examples of usage:* Unzip ZIP archive "archive.zip", then delete the archive.

```
{sapl:I sapl:do java:ubiware.shared.UnzipperBehavior}
sapl:configuredAs
{
      p:input sapl:is "archive.zip".
      p:delete sapl:is "true"
}
```

*Libraries needed:* none.

### 1.2.3 HttpDataFetcherBehavior

*Full name:* ubiware.shared.HttpDataFetcherBehavior

*Action:* Downloads a document (e.g. an HTML page) from an Internet server and saves it into either a file on the local file system or as a belief structure. The download process is done in a separate thread, so does not block the agent's lifecycle. If everything is fine, ends in success. If there is a problem with either fetching or saving, ends in failure.

*Inputs:*

| Name | Meaning | Range | Mandatory | Defaul |
|---|---|---|---|---|
| p:url | URL of the document. | String | Yes | |
| p:saveTo | Either the name of the file to save the document to, or a context container specifying the belief structure to be added (should use ?result variable that will be substituted with the result) | Filename or Context container | Yes | |

*Outputs:* Prints to the screen "Failed to retrieve <url>" if download failed.

Effect: Creates a file or specified belief structure.

*Examples of usage:* Download from the internet the file at location http://www.example.org/index.html

```
{sapl:I sapl:do java:ubiware.shared.HttpDataFetcherBehavior}
sapl:configuredAs
{p:url sapl:is
"http://www.fmi.fi/saa/paikalli.html?kunta=Jyv%E4skyl%E4".
p:saveTo sapl:is "received/weather.htm"
}

Download a file to the agent's beliefs structure:
{sapl:I sapl:do java:ubiware.shared.HttpDataFetcherBehavior}
     sapl:configuredAs
{p:url sapl:is
"http://www.fmi.fi/saa/paikalli.html?kunta=Jyv%E4skyl%E4".
p:saveTo sapl:is {sapl:I ex:haveWeatherKnowledge ?result}
}

Remarks: It is NOT so that using the ?result variable (as in the
second example) with a downloaded document that itself is S-APL
code, will result in the S-APL code being added as agent beliefs.
The contents of the file will be added as a string literal.
```

*Libraries needed:* none.

### 1.2.4 EmailSenderBehavior

*Full name:* ubiware.shared.EmailSenderBehavior

*Action:* Sends an email. The sending process is done in a separate thread, so does not block the agent. If the server responds with error 451 (local error in processing) or 452 (insufficient system storage), and parameter "waitOnFail" is given, waits specified time and tries again. If email is sent, ends in success. Ends in failure if an exception other than error 451 or 452 occurs or on any exception, if "waitOnFail" is not given. Also ends in failure if parameter "to" is not given or empty.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:smtp | SMTP server to use for sending email. | String (valid server address) | Yes | |
| p:smtpPort | The port used by the SMTP server. | Integer | No | 25 |
| p:smtpUser | Username for the SMTP server, if required. | String | No | No authentication |
| p:smtpPassword | Password for the SMTP server, if required. | String | No | No authentication |
| p:to | Addresses to which the email to be sent, divided by commas. | String (valid e-mail address) | Yes | |
| p:from | Address to put as the outgoing address. | String (valid e-mail address) | No | "no@addr.ess" |
| p:cc | Addresses to which a copy to be sent. | String (valid e-mail address) | No | "" |
| p:bcc | Addresses to which a blind copy is to be sent. | String (valid e-mail address) | No | "" |
| p:subject | Subject of the email. | String | No | "" |
| p:message | The content of the email. | String | No | "" |
| p:attach | Names of the files to attach to the email, divided by comma. | Filename(s) (separated by commas) | No | "" |

| | | | | |
|---|---|---|---|---|
| p:contentType | The type of the email content, e.g. “text/plain” or “text/html”. | String (valid content type) | No | “text/plain” |
| p:waitOnFail | Time to wait in milliseconds before next attempt of sending. | Integer | No | |

*Outputs:* On success, prints to the screen “Email to <to> is sent”. On failure, prints “Sending email to <to> is failed!”. In case of error 451 or 452, prints to the screen “Email sending error. Waiting <waitOnFail/1000> sec”.

Effect: the e-mail is send to the recipient.

*Examples of usage:* Send email through smtp.jyu.fi server. If a problem, wait 10 seconds and try again.

```
{sapl:I sapl:do java:ubiware.shared.EmailSenderBehavior}
     sapl:configuredAs
          {p:smtp sapl:is “smtp.jyu.fi”.
           p:to sapl:is “akataso@jyu.fi” .
           p:from sapl:is “akataso@cc.jyu.fi” .
           p:subject sapl:is “test” .
           p:message sapl:is ?text.
           p:waitOnFail sapl:is 10000
          }
```

*Libraries needed:* JavaMail (mail.jar) and JavaBeans Activation Framework (activation.jar). Both are included into J2EE, but need to be acquired separately for J2SE. (The libraries are provided with the UBIWARE platform.)

### 1.2.5 FilesSensorBehavior (deprecated)

*Full name:* ubiware.shared.FilesSensorBehavior

*Action:* Generates a belief structure describing the contents of a file-system folder, including all the subfolders.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:folder | Folder to sense. | Filename | Yes | |

*Effect:* Adds the following belief structure:

<folder> :folder {<subfolder> :folder { …}. <name> :file <extension>}

For subfolders and files; <name> is the full name of the file. (includes both the path with the start folder and the extension.)

*Remarks:* The behavior in its current form is deprecated, since it is using undefined default name-spacing. The effect of this behavior will be changed to one which uses namespaces in the future

*Examples of usage:* Sense the folder DB/_ontology

```
{sapl:I sapl:do java:ubiware.shared.FilesSensorBehavior}
     sapl:configuredAs {p:folder sapl:is DB/_ontology}.

{DB/_ontology :folder {?file :file ?ext}} => {...}
```

*Libraries needed:* none.

### 1.2.6 TextReaderBehavior

*Full name:* ubiware.shared.TextReaderBehavior

*Action:* Reads a text file. Creates a given belief structure with the contents of the file put as one of the resources.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:input | Name of the file. | Filename | Yes | |
| p:saveTo | A context container specifying the belief structure to be added (should use ?result variable that will be substituted with the file contents). | Context container | Yes | |

*Effect:* Adds the given belief structure.

Example of usage: read the content of the file "myfile.ext" to the agent's beliefs as the object of the statement ex:theFile ex:hasContent <content of the file>

```
{sapl:I sapl:do java:ubiware.shared. TextReaderBehavior }
     sapl:configuredAs
{
p:input sapl:is "myfile.ext" .
p:saveTo sapl:is {ex:theFile ex:hasContent ?result}
}.
```

### 1.2.7 TextWriterBehavior

*Full name:* ubiware.shared.TextWriterBehavior

*Action:* Writes a text string to a text file.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:input | The text to be written. | String | Yes | |
| p:saveTo | The name of the file. | Filename | Yes | |

*Effect:* Creates/overwrites a specified file.

Example of usage: Write the text “This text will be written to the file” to the file “myfile.ext”.

```
{sapl:I sapl:do java:ubiware.shared. TextWriterBehavior }
     sapl:configuredAs
{
p:input sapl:is “This text will be written to the file” .
p:saveTo sapl:is “myfile.ext”
}.
```

### 1.2.8 XmlReaderBehavior

*Full name:* ubiware.shared.XmlReaderBehavior

*Action:* Reads an XML file; then, generates a hierarchical belief structure (see below) in the agent's beliefs from its contents.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:input | Name of the file. | Filename | Yes | |

*Effect:* Adds the following belief structure:

<input> :tree {

{<elementNum> :tag <elementName>} :branch {

{<elementNum> :attribute <attributeName>} :leaf <attributeValue>.

<elementNum> :leaf <literalNode>.

{<elementNum> :tag <elementName>} :leaf :empty.

{<elementNum> :tag <elementName>} :branch {…}

}

}

<elementNum> is the consequent number of the element (regardless of type) inside its parent element. Leading zeros are added, so elementNum is always 4 digits, e.g "0001".

Remarks: reading and analyzing XML in S-APL might reduce the performance of your application and might be implemented more conveniently in a separate specialized RAB.

*Examples of usage:* Read the contents of the file contact.xml, having the structure as shown below. Then, find the phone number of the employee John Smith

<employees>

<contact-info>

<name>John Smith</name>

<phone>(212) 555-4567</phone>

</contact-info>

…

</employees>

```
{sapl:I sapl:do java:ubiware.shared.XMLReaderBehavior}
     sapl:configuredAs {p:input sapl:is contact.xml}.

{contact.xml :tree {
     {* :tag employees} :branch {
          {* :tag contact-info} :branch {
               {* :tag name} :branch {* :leaf "John Smith"}.
```

```
                {* :tag phone} :branch {* :leaf ?phone}}}}
} => {...?phone...}
```

*Libraries needed:* none (uses SAX API which is provided with Java SE).

### 1.2.9 XmlWriterBehavior

*Full name:* ubiware.shared.XmlWriterBehavior

*Action:* Writes XML data into a file or into a belief, based on a belief structure like that described above in XmlReaderBehavior.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:input | Context container that is the root of the sub-graph with XML data. | Context container | Yes | |
| p:saveTo | Either the name of the file to save the result to, or a context container specifying the belief structure to be added (should use ?result variable that will be substituted with the result). | Filename or Context container | Yes | |
| p:sort | If "true", the order of sub-elements of an element is defined by lexicographic ordering based on their <elementNum> (see above in XmlReaderBehavior). | Boolean | No | "true" |
| p:indents | If is equal to "true", the line breaks and indentation is used to separate XML elements. Otherwise, a single line is produced. | Boolean | No | "false" |
| p:encoding | If given, is put as the value of the "encoding" attribute in the XML header. | String | No | |

*Effect:* the XML file is written to disk or the specified belief structure is created.

Remarks: reading and analyzing XML in S-APL might reduce the performance of your application and might be implemented more conveniently in a separate specialized RAB.

*Examples of usage:* Save to "new.xml" the structure created in the example of XmlReaderBehavior.

```
{contact.xml :tree ?x} =>
 {{sapl:I sapl:do java:ubiware.shared.XmlWriterBehavior}
     sapl:configuredAs
          {p:input sapl:is ?x. p:saveTo sapl:is new.xml.
           p:indents sapl:is true}}
```

*Libraries needed:* none.

### 1.2.10 TextTableReaderBehavior

*Full name:* ubiware.shared.TextTableReaderBehavior

*Action:* Reads a text file containing a data-table, for example a CSV file (comma-separated-values). Generates a hierarchical belief structure (see below).

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:input | Name of the file. | Filename | Yes | |
| p:encoding | The name of the file encoding. "default" implies using the default system encoding. For encoding names consult *java.nio.charset.Charset.* Javadoc. For example, use "UTF-16" for Unicode. | String | No. | "default" |
| p:rowSeparator | A regular expression to use as separator for data rows. For information on regular expressions, consult the Java API docs about the class *java.util.regex.Pattern*. | String (regular expression) | No | "\\n" |
| p:columnSeparator | A regular expression to use as separator for data columns. | String (regular expression) | No | "," |
| p:selectColumns | A whitespace-separated list of the column indexes (one-based) to include. "all" implies including all the columns. For example "2 5 6 1" will select columns 2, 5, 6 and 1 (in that order). | String | No | "all" |

*Effect:* Adds the following belief structure:

<input> :table {<rowNum> :row { <columnNum>  :column <value> …} … }

Remark: Processing big files using this behavior might be inefficient since the whole file will be loaded in the agent's beliefs (=computer memory) before any processing can happen. If only a small piece of the data is needed, consider making a specialized RAB.

*Examples of usage:* Read the first and the second columns from Unicode (UTF-16) file "some.txt" where rows are separated by either comma or a whitespace. Then do something based on the result of the operation.

```
{sapl:I sapl:do java:ubiware.shared.TextTableReaderBehavior}
     sapl:configuredAs {p:input sapl:is "some.txt" .
                        p:columnSeparator sapl:is ",|\\s" .
                        p:encoding sapl:is "UTF-16" .
                        p:selectColumns sapl:is "1 2"}.
{"some.txt" :table {?rid :row {1 :column ?firstname.
                               2 :column ?lastname}
} } => {...}
```

*Libraries needed:* none.

### 1.2.11 ExcelReaderBehavior(deprecated)

*Full name:* ubiware.shared.ExcelReaderBehavior

*Action:* Reads data from a worksheet of a Microsoft Excel file. This behavior takes the first worksheet only, considers only columns that have something in the first row and considers that content as a label. For every row from the second on, generates a set of beliefs using those labels as predicates (see below for details). Ends in a failure if an I/O exception occurs.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:input | Name of Excel file | Filename | Yes | |

*Outputs:* Adds the following belief structure:

<input> :table {<rowNum> :row { <label>  :column <value> …} … }

This is done for every row from the second on. For every column that has something (<label>) in the first row, value is the value of the cell.

Remarks: This RAB is deprecated since it uses default name spaces. Furthermore, its behavior is considered too restrictive. If parsing of Excel documents is needed, consider implementing a specific RAB.

*Examples of usage:* Read data from file.xls, then do something based on that.

```
{sapl:I sapl:do java:ubiware.shared.ExcelReaderBehavior}
     sapl:configuredAs
          {p:input sapl:is file.xls}.


{ file.xls :table {?id :row {Name :column ?name.
                             Email :column ?email.
                             Saldo :column ?saldo}
} } => {...}
```

*Libraries needed:* Jakarta POI by Apache (poi.jar). Acquired from http://jakarta.apache.org/poi/. Provided with the UBIWARE platform.

### 1.2.12 SQLReaderBehavior

*Full name:* ubiware.shared.SQLReaderBehavior

*Action:* Submits an SQL query to a relational database and generates from the results a hierarchical belief structure (see below).

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:driver | The name of the class of the database driver, e.g. oracle.jdbc.OracleDriver or sun.jdbc.odbc.JdbcOdbcDriver | String | Yes | |
| p:url | The URL of the database | String | Yes | |
| p:username | The username for the database connection. | String | No | “” |
| p: password | The password for the database connection. | String | No | “” |
| p:query | The SQL query to submit | String | Yes | |

*Outputs:* Adds the following belief structure:

<url> :table {<rowNum> :row { <columnName>  :column <value> …} … }

*Examples of usage:* Read all the data from the MEMBERS table of an MS Access database having ID “MemberRegister” in ODBC.

```
{sapl:I sapl:do java:ubiware.shared.SQLReaderBehavior}
     sapl:configuredAs
          {p:database sapl:is "sun.jdbc.odbc.JdbcOdbcDriver".
           p:url sapl:is "jdbc:odbc:MemberRegister".
           p:username sapl:is "Admin".
           p:password sapl:is "blablabla".
           p:query sapl:is "SELECT * FROM MEMBERS"}.

{jdbc:odbc:MemberRegister :table
               {?rid :row {FirstName :column ?firstname.
                           LastName :column ?lastname}
} } => {...}
```

*Libraries needed:* none. However, making connection to a specific database may require a library with the specific drivers.

### 1.2.13 RdfXmlReaderBehavior

*Full name:* ubiware.shared.RdfXmlReaderBehavior

*Action:* Reads am RDF/XML file and loads its contents into the agents' beliefs into the specified context container.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:input | The name of the input file. | Filename | Yes | |
| p:saveTo | ID of context container, which is to be the root of a newly created sub-graph. If "G" is given, the beliefs are added to the general (top-most) context. Has to be given as a typed literal to avoid treating as an actual container (see example). | Context container as String!!! | No | "G" |

*Outputs:* none.

*Examples of usage:* Load the contents of the file airport.owl into the container {} :accordingTo :AirportOntology.

```
{? ? ?} :accordingTo :AirportOntology.

{?target :accordingTo :AirportOntology} =>
     {{sapl:I sapl:do java:ubiware.shared.RdfXmlReaderBehavior}
          sapl:configuredAs
               {p:input sapl:is tests/test.owl.
                p:saveTo sapl:is "?target"^^xsd:string.
                sapl:End sapl:remove{
                    {? ? ?} :accordingTo :AirportOntology
                }
               }
     }
```

*Libraries needed:* Sesame 2.0 and some its utilities (openrdf-sesame-2.0-onejar.jar, slf4j-api-1.3.0.jar, slf4j-jdk14-1.3.0.jar). Acquired from http://www.openrdf.org/. Provided with the UBIWARE platform.

### 1.2.14 BeliefsBackupBehavior

*Full name:* ubiware.shared.BeliefsBackupBehavior

*Action:* Saves a sub-graph of agents' beliefs as an S-APL Notation3 string into a file or as a literal into a belief.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:input | ID of context container that is the root of the sub-graph. If "G" is given, the whole beliefs storage is saved. | Context container | No | "G" |
| p:saveTo | Either the name of the file to save the result to, or a context container specifying the belief structure to be added (should use ?result variable that will be substituted with the result). | Filename or Context container | Yes | |
| p:indents | If is equal to "true", the line breaks and indentation is used to separate beliefs. Otherwise, a single line is produced. | Boolean | No | "false" |

*Effect:* the file with Notation3 representation of beliefs will be saved to the disk or a copy of the beliefs will be stored in the agent's beliefs' structure.

Remarks: Before using this behavior, consider using the persistency infrastructure.

*Examples of usage:* Save to "goals.sapl" the contents of sapl:I sapl:want {} container, i.e. all the active goals of the agent.

```
{sapl:I sapl:want ?in} =>
     {{sapl:I sapl:do java:ubiware.shared.BeliefsBackupBehavior}
      sapl:configuredAs {p:input sapl:is ?in.
                         p:saveTo sapl:is "goals.sapl"}
```

*Libraries needed:* none.

### 1.2.15 BeliefsLoadBehavior

*Full name:* ubiware.shared.BeliefsLoadBehavior

*Action:* Loads an S-APL Notation3 string (given directly or from file) into the agents' beliefs.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:input | The S-APL Notation3 string. (must be valid s-apl code as such) It can for example not make use of the prefixes defined in the surrounding s-apl context) | String | Yes, if p:inputFromFile is not given. | |
| p:inputFromFile | The name of the file with S-APL Notation3. | Filename | Yes, if p:input is not given. | |
| p:saveTo | ID of context container, which is to be the root of a newly created sub-graph. If "G" is given, the beliefs are added to the general (top-most) context. Has to be given as a typed literal to avoid treating as an actual container (see example). | Context container as String!!!! | Yes | |
| p:doChecks | If is false, checks are not performed for joining containers. | Boolean | No | "true" |

*Outputs:* none.

Remark: consider using multiple scripts for loading beliefs on agent initialization before using this RAB.

*Examples of usage:* Load the contents of the file "goals.sapl" into sapl:I sapl:want {} container.

```
{sapl:I sapl:want ?c} =>
     {{sapl:I sapl:do java:ubiware.shared.BeliefsLoadBehavior}
      sapl:configuredAs {p:saveTo sapl:is "?c"^^xsd:string.
                         p:inputFromFile sapl:is "goals.sapl"}
```

*Libraries needed:* none.

## 1.3 Agent HTTP Server actions

The ubiware agent can be used to handle HTTP requests. In order to do so, the agent can start an HTTP server by using the AgentServerBehavior. After this is done, any HTTP request arriving on to the server will be put on the blackboard of the agent under the form of an AgentServerBehaviorServerEvent. This object contains a HttpServletRequest, a HttpServletResponse, and an org.eclipse.jetty.continuation.Continuation object. These objects can then in turn be used to answer to the request by for example using the HttpResponseSenderBehavior. Another way to give agents web functionality is using the UBIWARE Infrastructure. This might be more complicated for simple applications, but greatly simplifies development for more complicated applications.

### 1.3.1 AgentServerBehavior

*Full name:* ubiware.shared.agentServer.AgentServerBehavior

*Action:* Start an http server on the specified port.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:port | The port on which the server should be running. | Integer | Yes | |
| p:timeOut | The timeout in milliseconds for the agent to respond to the request. After this time, the Continuation will time out. | Integer | No | `300000 (=5 min)` |

*Implicit inputs:* None.

*Outputs:* None except for debug output from underlying Jetty implementation.

*Effect:* The server is started on the specified port. When a request arrives to the server, an object is added to the blackboard containing all needed information to answer the request. This object (of type AgentServerBehaviorServerEvent) contains a HttpServletRequest, a HttpServletResponse, and an Continuation object. Next to this, a belief about the arrival of the request is added to the agent's beliefs' structure. These beliefs have the following structure:

```
<requestID> sapl:hasBlackboardID <blackboardID> ; sapl:requestsPath <path>

Where,
```

- <requestID> is a random ID in namespace http://www.ubiware.jyu.fi/id#
- <blackboardID> is a random id with which the event's associated object can be found from the blackboard (an AgentServerBehaviorServerEvent)
- <path> is as defined in HttpServletRequest getPathInfo()but with the first '/' stripped of!

*Remarks:*

Answering to request can be done using HttpResponseSenderBehavior.

This behavior functions in a slightly different way as its predecessors. There are ways to mimic the behavior of the old versions of this behavior but these are not supported. (see for example the use of p:obsoleteMode in the behavior code).

*Examples of usage:* start an HTTP server on port 5000 and give a default response timeout of 2 minutes.

```
{sapl:I sapl:do java: ubiware.shared.agentServer.AgentServerBehavior }
sapl:configuredAs
{
     p:port sapl:is 5000 .
     p:timeOut sapl:is 120000
}
```

*Libraries needed:* All libraries needed to run the Jetty web server.

### 1.3.2 HttpResponseSenderBehavior

*Full name:* ubiware.shared.HttpResponseSenderBehavior

*Action:* Sends an HTTP response to a request arrived earlier.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:message | The response to send. | String | Yes, if p:messageFromFile is not given | |
| p:messageFromFile | The name of the file, from which the content of the response is to be loaded. | Filename | Yes, if p:message is not given | |
| p:socket | The ID of the blackboard object | ID of a blackboard object of type AgentServerBehaviorServerEvent | Yes | |

*Effects:* sends HTTP response and removes the object from the agent's blackboard.

*Remarks:* Using this behavior does only allow you to give basic responses to requests. It does for example not allow usage of Http headers (cookies, etc). Furthermore, S-APL is probably not the best language to generate dynamic web content. When more complicated web-pages are required, consider writing your own version of this behavior or use the Infrastructure agents provided by the UBIWARE platform for handling web requests.

*Examples of usage:* Send the line "bla bla" is response to the request asking for "what.html" and then delete the request.

```
{
        ?requestID sapl:hasBlackboardID ?blackboardID ;
                    sapl:requestsPath "what.html"
}
=>
{
        {sapl:I sapl:do java:ubiware.shared. HttpResponseSenderBehavior}
        sapl:configuredAs
        {
                p:message sapl:is "bla bla" .
                p:socket sapl:is ?blackboardID .
                sapl:End sapl:remove {
                        ?requestID * *
                }
        }
}
```

*Libraries needed:* All libraries needed to run the Jetty web server.

## 1.4 Agent GUI

The UBIWARE agent has a very limited support for a local GUI for the agent. The preferable way to make GUI's is using HTML web pages. A local GUI might be of use for Debugging or basic agent programming. Only one simple GUI component is provided with the UBIWARE platform. Writing more complicated components is possible, but controlling the whole components from the agent might become cumbersome. The code in the package ubiware.shared.gui.buttonGUI can be used as an example. Keep in mind that this GUI is not compatible with agents' mobility.

### 1.4.1 NewGUIBehavior

*Full name:* ubiware.shared.gui.buttonGUI.NewGUIBehavior

*Action:* Create and display a new simple GUI which contains of a column of buttons.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:button | Give the name of the button. Specifying this parameter multiple times will result in the GUI having multiple buttons. Keep in mind that specifying 2 buttons with the same name will due to agent beliefs merge result in the creation of only one button. The name of the button is used as the text displayed on the button and for later reference. | String | Yes | |
| p:ID | The ID of this GUI, used for later reference. | Resource | No | <Random> |
| p:title | The title the window should have. | String | No | <Agent Name> |
| p:text | The text which accompanies the buttons. | String | No | "" |
| p:closable | Is the GUI closable by the user? If set to true, the GUI will disappear as soon as the close button is clicked. Otherwise, only the agent is able to close the GUI by using disposeGUIBehavior. | Boolean | No | "true" |

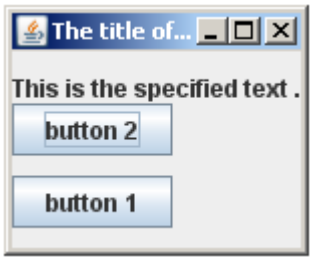
*Implicit inputs:* None

*Outputs:* None

*Effect:* The GUI will be shown on the screen connected to the host where the agent calling the behavior resides. Clicking of buttons will result in beliefs added to the UBIWARE agent.

*Remarks:* The GUI only works locally and is incompatible with agent's mobility.

*Examples of usage:* The following code results in GUI as shown in the picture.




```
{sapl:I sapl:do
java:ubiware.shared.gui.buttonGUI.NewGUIBehavior }
sapl:configuredAs
{
      p:button sapl:is  "button 1" .
      p:button sapl:is "button 2" .
      p:title sapl:is "The title of the window." .
      p:text sapl:is "This is the specified text ." .
      p:ID sapl:is ex:IDOftheGUI .
      p:closable sapl:is "false" .
} .
```

Notice that it is not possible to define the order of the buttons since by definition, an agent's context container does not have any order.

When the users clicks any of the buttons of the GUI, a belief of the following form will appear in the agent's beliefs'structure:

```
{
    :User :clickedButton <name of the button>
} sapl:eventFromGUI <the ID of the GUI>
```

concrete, if the user clicks the button name "button1", the following belief appears:

```
{
    :User :clickedButton "button 1"
} sapl:eventFromGUI ex:IDOftheGUI
```

The agent should then have rules implemented to handle these events.

When the user tries to close the GUI (independent on the closable parameter), the agent gets the following belief:

```
{
    :User :clickedFrameButton "close"
} sapl:eventFromGUI ex:IDOftheGUI
```

*Libraries needed:* None

### 1.4.2 DisposeGUIBehavior

*Full name:* ubiware.shared.gui.buttonGUI.DisposeGUIBehavior

*Action:* Cleans up a Button GUI created with NewGUIBehavior.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:ID | The ID of the GUI which should be disposed. | String | Yes | |

*Effect:* The GUI gets disposed. This behavior ends in a failure if there is no such GUI.

*Remarks:* Referencing to the same GUI ID after disposal of the GUI will result in failures.

*Examples of usage:* Imagine we have a GUI as created by the example code in NewGUIBehavior. Now we will dispose this GUI.

```
{sapl:I sapl:do java:ubiware.shared.gui.buttonGUI.DisposeGUIBehavior }
sapl:configuredAs
{
      p:ID sapl:is "IDOftheGUI" .
} .
```

*Libraries needed:* None.

### 1.4.3 DisableButtonBehavior

*Full name:* ubiware.shared.gui.buttonGUI.DisableButtonBehavior

*Action:* Disable a button in a GUI if it was previously enabled. do nothing otherwise.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:ID | The ID of the GUI | String | Yes | |
| p:target | The button which is target of the disabling action. This is the same as the String specified for the p:button parameter specified when starting the GUI. | String | Yes | |

*Outputs:* None

*Effect:* The specified button is disabled in the specified GUI.

*Remarks:* Use this behavior only after the GUI is already displayed. For initial disabling of buttons, put the call for this behavior in the sapl:Success sapl:add part of the NewGUIBehavior.

*Examples of usage:* We will disable the button “button 2” from the GUI created in the example of NewGUIBehavior. Enabling the button again is possible by using EnableButtonBehavior.

```
{sapl:I sapl:do
java:ubiware.shared.gui.buttonGUI.DisableButtonBehavior }
           sapl:configuredAs
           {
                p:ID sapl:is "IDOftheGUI" .
                p:target sapl:is "button 2"
           }
```

*Libraries needed:* None

### 1.4.4 EnableButtonBehavior

*Full name:* ubiware.shared.gui.buttonGUI.EnableButtonBehavior

*Action:* enable a button on the GUI if it was previously disabled. Do nothing otherwise.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:ID | The ID of the GUI | String | Yes | |
| p:target | The button which is target of the enabling action. This is the same as the String specified for the p:button parameter specified when starting the GUI. | String | Yes | |

*Outputs:* None

*Effect:* The specified button is enabled in the specified GUI.

*Remarks:* None

*Examples of usage:* We will enable the button "button 2" from the GUI created in the example of NewGUIBehavior that could previously have been disabled with DisableButtonBehavior.

```
{sapl:I sapl:do java:ubiware.shared.gui.buttonGUI.EnableButtonBehavior
}
           sapl:configuredAs
           {
                p:ID sapl:is "IDOftheGUI" .
                p:target sapl:is "button 2"
           }
```

*Libraries needed:* None

## 1.5 Inter-agent actions

This section describes behaviors that involve other agents, such as communicative actions.

### 1.5.1 CreateAgentBehavior

*Full name:* ubiware.shared.CreateAgentBehavior

*Action:* Creates a new agent in the same agent container.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:name | The name of the agent. | Resource | Yes | |
| p:scripts | Scripts of the agent. Star(*) separated. | String | No | “” |
| p:roles | Roles of the agent. Plus(+) separated. Needed if startup.sapl model is used. | String | No | “” |
| p:initBeliefs | Beliefs which should initially be added to the created agent. | Context container | No | “” |

Effect: The specified agent is created.

*Remark:* It is possible to create a zombie agent this way. Make sure at least one of the parameters p:scripts or p:initBeliefs is specified. Make sure the agent name is unique in the platform. The platform does not allow two agents with the same name to be started.

*Examples of usage:* Start an agent with initial believes sapl:I ex:createdBy <creating agent>

```
{sapl:I sapl:haveName ?myName}
=>
{
      {sapl:I sapl:do java:ubiware.shared.CreateAgentBehavior}
      sapl:configuredAs
      {
            p:name sapl:is ex:newAgent .
            p:initBeliefs sapl:is {
                  sapl:I ex:createdBy ?myName
            }
      }
}
```

Start an agent with initial script “myscript.sapl”

```
{sapl:I sapl:do java:ubiware.shared.CreateAgentBehavior}
      sapl:configuredAs
{
p:name sapl:is ex:newAgent .
      p:scripts sapl:is "myscript.sapl"
}
```

*Libraries needed:* none.

### 1.5.2 MessageSenderBehavior

*Full name:* ubiware.shared.MessageSenderBehavior

*Action:* Sends a message to another agent on the same platform.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:receiver | Name of the agent to send message to. | Resource | Yes | |
| p:content | Content of the message. | String or context container. Only context container allowed if p:contentIsSAPL is set to "true" | Yes | |
| p:conversationID | ID of the conversation. If not given, a new one is generated. | String | No | Same as the messageID |
| p:performative | ACL performative for the message, like for example "request", "inform", … | `"accept-proposal" ,"agree" ,"cancel" ,"cfp" ,"confirm" ,"disconfirm" ,"failure" ,"inform" ,"inform-if" ,"inform-ref" ,"not-understood" ,"propose" ,"query-if" ,"query-ref" ,"refuse" ,"reject-proposal" ,"request" ,"request-when" ,"request-whenever" ,"subscribe" ,"proxy" ,"propagate"` | No | "request" |
| p:ontology | The ontology of the message. | String | No | "unspecified" |

| | | | | |
|---|---|---|---|---|
| p:addBeliefs | Whether the agent is to have memory of sending this message. Can be either “false”, a context with a pattern, or “true” (use of the predefined pattern). See Effect for more details. | Boolean | No | “true” |
| p:contentIsSAPL | If “false”, the content is treated as a literal; if “true”, as an S-APL graph. | Boolean | No | Determined automatically. If given, checked before sending. |
| p:cleanContent (deprecated) | If “true”, the content is cleaned in the sense that potential query constructs such as *Optional*, *or*, filtering are not included. | Boolean | No | “false” |
| p:print | Controls whether anything is printed to the screen when a message is sent. | Boolean | No | “false” |

*Outputs:* A message indicating the sending of the message is printed to the console if p:print is set to true.

*Effects:* The message is send to recipient. A random messageID is generated.

If p:addBeliefs is equal to “true”, the following belief structure is added:

```
<messageID>  p:sent {
p:conversationID  sapl:is <value of p:conversationID> .
p:performative sapl:is <value of p:performative> .
p:receiver sapl:is < value of p:receiver> .
p:ontology sapl:is < value of p:ontology>.
p:content sapl:is < value of p:content>
}
```

Depending on p:contentIsSAPL, < value of p:content> is either a literal or a context container.

“Alternatively, p:addBeliefs can be given an container with a pattern, which can refer to the following variables: ?messageID, ?conversationID, ?receiver, ?performative, ?ontology, ?content. Then, instead of the predefined pattern above, the given one is added while substituting the parameters for their respective values.

*Remarks:* Keeping p:addBeliefs to its default value might imply major memory consumption since all content referred to by the message will be kept in the agent’s memory. The use of the

parameter p:cleancontent is discouraged and the parameter is deprecated. Enabling printing of message sending is only intended for debugging and demonstration purposes, do not leave it enabled in production systems.

Sending literal strings is discouraged.

*Examples of usage:* Send the message “ex:Jane ex:loves ex:John” to ex:newAgent with performative query-if.

```
{sapl:I sapl:do java:ubiware.shared.MessageSenderBehavior}
sapl:configuredAs
{
     p:receiver sapl:is ex:newAgent .
     p:ontology sapl:is “love”.
     p:performative sapl:is “query-if” .
     p:content sapl:is { ex:Jane ex:loves ex:John }
}
```

*Libraries needed:* none.

### 1.5.3 QueryMessageSenderBehavior

*Full name:* ubiware.shared.queryMessageSenderBehavior

*Action:* sends a message which content is specified by a query.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:query | The query which should be performed. | Context container | Yes | |
| p:context | The context container into which the query must be performed. | Context container | Yes | |
| p:preamble | A container of S-APL code which will be added to the message (even if the query resulted in no matches) | Context container | No | none |
| p:receiver | The receiving agent | Resource | Yes | |
| p:sendPattern | The pattern which will be used for sending. Variables will be bound from variables specified in the p:query parameter using simple substitution. | context container | No | Same as the value assigned to p:query |
| p:conversationID | ID of the conversation. If not given, new one is generated. | String | Yes | |
| p:performative | ACL performative for the message, like for example “request”, “inform”, … | `"inform", "query-if", "request"` | Yes | |
| p:ontology | The ontology of the message. | String | No | “unspecified” |

*Implicit inputs:* The behavior performs a query in the specified container.

*Outputs:* None.

*Effect:* A message is send to the receiver with the following content:

1) The preamble
2) For every match of the query specified by p:query in the context specified by p:context, the variables are bound and replaced in the p:sendPattern. the result of this is appended.

Further parameters have the same effect as in normal MessageSenderBehavior.

*Remarks:* Adding beliefs and printing is supposed to be done in the success part of this RAB. That is why those parameters are not available.

*Examples of usage:* Assume we have the following beliefs in our agent:

```
ex:weatherInfo ex:desrcibedBy {
      ex:temperature ex:is 45 .
      ex:wind ex:hasDirection ex:N .
```

```
      ex:wind ex:hasSpeed 100 .
      ex:climate ex:is ex:type25 .
      ex:wind2 ex:hasDirection ex:S .
      ex:wind2 ex:hasSpeed 200 .
      ex:wind3 ex:hasDirection ex:N .
      ex:wind3 ex:hasSpeed 120 .
}
```

And we want to send a message containg all wind speeds of wind having direction ex:N. then we use the following RAB call:

```
{
      ex:weatherInfo ex:describedBy ?queryContainer
}
=>
{
      {sapl:I sapl:do java:ubiware.shared.QueryMessageSenderBehavior}
      sapl:configuredAs
      {
            p:query sapl:is {
                  ?wind ex:hasSpeed ?speed .
                  ?wind ex:hasDirection ex:N
            } .
            p:context sapl:is ?queryContainer .
            p:preamble sapl:is {
                  ex:service ex:hasStatus ex:running
            } .
            p:receiver sapl:is ex:newAgent .
            p:sendPattern sapl:is {
                  ?wind ex:hasSpeed ?speed
            } .
            p:conversationID sapl:is "conversation-456" .
            p:performative sapl:is "inform"
      }
}
```

*Which will result in the following message content being send:*

```
@prefix ex: <http://www.example.com/ubiwareexamples#> .
ex:wind ex:hasSpeed "100" .
ex:wind3 ex:hasSpeed "120" .
ex:service ex:hasStatus ex:running
```

*Libraries needed:* None

### 1.5.4 MessageReceiverBehavior

*Full name:* ubiware.shared.MessageReceiverBehavior

*Action:* Listens for incoming message(s) matching defined parameters (or all messages if none given). Upon receiving a matching message, generates a belief about this (only if "addBeliefs" is not equal to "false") and wakes the agent. If "waitOnlyFirst" is equal to "true", ends in success after receiving one message. If "maxWait" is given and the specified period expires, ends in failure. Otherwise, continues receiving messages forever.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:matchConversationID (>1 allowed) | Match conversation ID. If several is given, connected with logical OR (the same applies to all below). | String | No | |
| p:matchPerformative (>1 allowed) | Match ACL performative, like for example "request" , "inform", … | `"accept-proposal" ,"agree" ,"cancel" ,"cfp" ,"confirm" ,"disconfirm" ,"failure" ,"inform" ,"inform-if" ,"inform-ref" ,"not-understood" ,"propose" ,"query-if" ,"query-ref" ,"refuse" ,"reject-proposal" ,"request" ,"request-when" ,"request-whenever" ,"subscribe" ,"proxy" ,"propagate"` | No | |
| p:matchOntology (>1 allowed) | Match the message ontology. | String | No | |
| p:matchSender (>1 allowed) | Match the local name of the message sender. | Resource | No | |
| p:matchContent (>1 allowed) | Match the content of the message (as string, can thus not be used to match S-APL contnent) | String | No | |
| p:waitOnlyFirst | Whether to close the receiver after the first matching message is received. | Boolean | No | "true" |
| p:maxWait | Time in milliseconds after which receiver must close. | Integer | No | infinite |

| | | | | |
|---|---|---|---|---|
| p:addBeliefs | Whether the agent is to have memory of receiving this message. Can be either “false”, a context with a pattern, or “true” (use of the predefined pattern). | Boolean or Context container | No | “true” |
| p:contentIsSAPL | If “false”, the content is treated as a literal; if “true”, as an S-APL graph. This affects only the addBeliefs procedure. | Boolean | No | “true” |
| p:contentToFile | The name of the file where the content of the message is to stored. | File | No | |
| p:print | Controls whether anything is printed to the screen when a message arrives. Can be either “true”, “false” or an integer representing nr of characters of the message are displayed. | Boolean or Integer | No | “200” |

*Outputs:* Prints to the screen “"<content>” from <sender> received”. If “addBeliefs” is equal to “true”, a message ID is generated randomly.

Effect: The following belief structure is added (objects of triples come from the received message):

```
<messageID> p:received {
      p:conversationID  sapl:is <conversationID> .
      p:performative sapl:is <performative> .
      p:sender sapl:is <sender> .
      p:ontology sapl:is <ontology>.
      p:content sapl:is <content>
}
```

Depending on p:contentIsSAPL, <content> is treated as either a literal or a context container.

p:addBeliefs can be given a container with a pattern, which may refer to the following variables: ?messageID, ?conversationID, ?sender, ?performative, ?ontology, ?content. Then, instead of the predefined pattern above, the given one is added with the parameters substituted.

*Examples of usage:* Receive messages with ontology “love” for 1 minute (while adding the predefined pattern of beliefs for every request received). After 1 minute, end and add belief “sapl:I ex:Ended ex:Receiving”.

```
{sapl:I sapl:do java:ubiware.shared.MessageReceiverBehavior}
sapl:configuredAs
{
      p:matchOntology sapl:is “love”.
      p:matchPerformative sapl:is “query-if”.
      p:waitOnlyFirst sapl:is false.
      p:maxWait sapl:is 60000.
      sapl:Fail sapl:add {sapl:I ex:Ended ex:Receiving}
}
```

*Libraries needed:* none.

### 1.5.5 DeleteAgentBehavior

*Full name:* ubiware.core.behaviors.DeleteAgentBehavior

*Action:* Kill the specified agent.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:name | The agent which should be stopped | Resource | Yes | |

*Implicit inputs:* None

*Outputs:* None

*Effect:* If the agent exists, this agent will be killed.

*Remarks:* Use this behavior with caution. If the agent being killed had still outstanding tasks, they will never be executed.

*Examples of usage:* Stop the agent with name ex:newAgent

```
{sapl:I sapl:do java:ubiware.core.behaviors.DeleteAgentBehavior}
sapl:configuredAs
{p:name sapl:is ex:newAgent}
```

*Libraries needed:* None

## 1.6 Agent Mobility

### 1.6.1 MoveBehavior

*Full name:* ubiware.shared.mobility.MoveBehavior

*Action:* Move the agent to a different container.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| p:destination | The destination container the agent should move to. | String | Yes | |

*Implicit inputs:* None

*Outputs:* Moving to location <p:destination> on the console of the source container.

*Effect:* The agent moves to the specified container.

*Remarks:* Moving is only possible if all of the agents blackboard objects are serializable and the agent is not performing any tasks local to the underlying platform like a server or local GUI.

Agent mobility is not guaranteed.

*Examples of usage:* Move the agent to container with name “secondary container”

```
{sapl:I sapl:do java:ubiware.shared.mobility.MoveBehavior}
sapl:configuredAs
{p:destination sapl:is “secondary container”}
```

*Libraries needed:* None.

## 1.7 Behavior Template

This section contains a template which can be used to describe the working of an RAB.

### 1.7.1 The name of the behavior, normally the java class name.

*Full name:* the fully qualified name of the behavior. The name the agent must specify to start it from inside its beliefs.

*Action:* a short description of the actions performed.

*Inputs:*

| Name | Meaning | Range | Mandatory | Default |
|---|---|---|---|---|
| The fully qualified name of the parameter. Can use defined prefixes. | The meaning of the parameter; how it is used, how it affects the action of this behavior. | The set of possible values accepted for this behavior. | 'Yes' if this parameter must be specified. 'No' otherwise. | The default value of this parameter if Mandatory is 'No' |

*Implicit inputs:* Which inputs does the behavior implicitly take from the agent. Like for example beliefs which are read from the agent which are not mentioned as parameters.

*Outputs:* Any output printed to the standard output and standard error by this behavior.

*Effect:* The effect this behavior has on the environment of the agent and its own belief structure. This is similar to what is described under Action, but more elaborated and precise.

*Remarks:* Remarks concerning the usage of this Behavior; why is it deprecated, which other behaviors are related and could perhaps be used instead, how is the performance of this behavior and so on.

*Examples of usage:* A concrete example on how to use this RAB. This example should be as essential as feasible. I.e. it should not contain more S-APL code as necessary.

*Libraries needed:* List of Java libraries used by this behavior and whether they are distributed with the standard platform distribution.