

# Process Enactment

## 1. Overview

Process enactment is what you do day-to-day, when carrying out your work.

Process enactment task types are described [below](#).

## 2. Process Enactment Task Types

### 2.1. Access Relational Database

#### 2.1.1. Description

Execute a query, insert or update SQL statement on a relational database at a specified URL. If the statement is a query, the result set is parsed into a predefined entity within the Role.

Parameters can be assigned to the SQL statement via bind variables, whose values are taken from specified entity components within the Role:

- An entity parameter is [converted to standard XML](#) for assignment to a bind variable
- An entity attribute is used as the String form of its value

In order for this task to succeed, a JDBC driver class for the database in question must be on the RADRunner classpath. The class to be used can optionally be specified using a system property of format `com.rolemodellers.jdbc.firstPartOfJdbcUrl=className`. For instance, with a database URL of format `mysql://guthrie:3306/metrics` you could specify system property `com.rolemodellers.jdbc.mysql=com.mysql.jdbc.Driver`. If no such property is set, RADRunner will attempt to determine a suitable driver class itself.

[How to add to a Role type](#)

#### 2.1.2. Parameters

None.

### 2.2. Add User

### 2.2.1. Description

Add a new user to the system.

[How to add to a Role type](#)

### 2.2.2. Parameters

**Property name**

The nickname by which the user is to be known in the current Role

**Object name**

The user's name [optional - if not supplied, property name will be used as the default value]

**Email**

The user's email address

**Password**

The user's logon password

**Confirm Password**

The user's logon password

**Machine**

Is the user to be fully automated, i.e., not for human logon?

**Offline**

Is the user's work to be carried out by a different system (a separate RADRunner installation, another process engine, or manually)?

## 2.3. Change User Of A Role Instance

### 2.3.1. Description

Assign a different user to a Role instance.

[How to add to a Role type](#)

### 2.3.2. Parameters

**Property name**

A Role instance

**User**

The new user of the Role instance

## 2.4. Computation

## *Process Enactment*

### **2.4.1. Description**

Do a calculation or transformation on data within the Role.

A RADRunner computation:

1. Takes a left-hand side operand
2. Takes an operation
3. For a binary operation, takes a right-hand side operand
4. Performs the operation
5. Uses the result to update the left-hand side operand.

The full operation list is:

- And
- Append
- Decrement
- DivideBy
- FractionalPart
- Increment
- Lower
- Minus
- MinusDays
- MinusHours
- MinusMilliseconds
- MinusMinutes
- MinusSeconds
- MinusWeeks
- Modulo
- MultiplyBy
- Nand
- Negate
- Not
- Now
- Or
- Plus
- PlusDays
- PlusHours
- PlusMilliseconds
- PlusMinutes
- PlusSeconds
- PlusWeeks

- Prepend
- Round
- SetTo
- SetToNull
- Truncate
- Upper
- Xor

The *left-hand side* of a computation must be an attribute value. The *right-hand side* can be a constant value, an attribute value, or a *pseudo-attribute* value (the number of attributes and sub-entities within an entity, or the sum of all numeric attributes within an entity and its sub-entities).

Any desired calculation or transformation on data can be effected by chaining together a sequence of computations. However, for certain very complex manipulations another approach may be easier to implement. For instance, numeric calculations could be simply effected using function calls in an [SQL select statement](#) on a dummy table, and text processing is better handled via [XSL transformation](#). Such approaches also have the advantage that the computation itself is defined externally to the Role (as an SQL statement or XSL file); thus it can not only be maintained in a separate repository if required, but can also be updated dynamically via an interaction (to replace the SQL statement or point to a different XSL file).

[How to add to a Role type](#)

#### 2.4.2. Parameters

None.

### 2.5. Convert User To Entity

#### 2.5.1. Description

Add the details of a specified user to the Role resources, by placing them in a specified entity.

The user can be specified either at define-time or at runtime:

1. If the task is non-automated and a user is specified at runtime, that one is used
2. Otherwise, if a user was specified in the task at define-time, that one is used
3. Otherwise, the user of the current Role is used.

This task can be used to get the email address of a user into a Role, for use in such tasks as [Send Email](#). In conjunction with [Create Reference To User](#), it can also be used to transfer

## Process Enactment

knowledge of a user between Roles, so as to delegate the ability to change their details, start and stop their Role instances, and so on.

### [How to add to a Role type](#)

#### 2.5.2. Parameters

##### **User**

The user to convert

## 2.6. Create Reference To User

### 2.6.1. Description

Create a new reference to an existing user within the Role.

To use this task, the id of the user must exist as an entity attribute value. The property name (nickname) of the user is also taken from an entity attribute value. The entity attributes can be specified either at define-time or at runtime - if both are specified, the runtime value(s) will be used.

In conjunction with [Convert User To Entity](#), this task can be used to transfer knowledge of a user between Roles, so as to delegate the ability to change their details, start and stop their Role instances, and so on. The Role which created (or imported) the user executes a task of type *Convert User To Entity* to store all details of the user within a Role resource. The user's id - and any other details required - can then be transferred to another Role via an interaction. This second Role executes *Create Reference To User* to define the user for themselves, using the id sent to them and any nickname they wish (the most common choice would be the same nickname used by the first Role).

*The above procedure should be used with care.* As soon as more than one Role knows about a user, there is a possibility that one Role will [delete the user](#) without informing the other Roles. This would not only cause inconsistency, but may also lead to process errors - for instance, if an attempt is made to assign a deleted user to a Role instance, it will fail.

### [How to add to a Role type](#)

#### 2.6.2. Parameters

##### **User**

The nickname of the user once a reference to it has been added to the Role

##### **Id**

An entity attribute containing the id of the user

## 2.7. Delete Users

### 2.7.1. Description

Delete users from the system.

[How to add to a Role type](#)

### 2.7.2. Parameters

#### Users

The user(s) to be deleted

## 2.8. Download File

### 2.8.1. Description

Download a file to the user's machine.

This task offers the user the opportunity to download to the client a specified file on the server. The user is presented with a prompt, explaining what they are about to download and why. They can then click on a link to open the file in a new browser window, from where the browser menu can be used to save it to their own machine.

Both prompt and filename are taken from resource attributes in the Role instance, specified at define time.

The original file is left on the server. An option to delete the file from the server is *not* provided - this could lead to loss of data, if the transfer to the client was corrupted for some reason or the data was lost once received. It is the system administrator's responsibility periodically to [clear from the server files that have become obsolete](#).

The task [External Component](#) can be used to view server files from within a web browser before download.

In conjunction with [Upload File](#), this task can be used not only to provide users with the means to download standard files from the server, but also to implement *web-based document management* via RADRunner: structured file transfer and maintenance by multiple users. Suppose, for example, that one role instance uploads a file via [Upload File](#). If the resource attribute containing the URL to the uploaded file is passed to another role instance via an interaction, the second role instance can then download the same file to a client via a *Download File* task.

[How to add to a Role type](#)

**2.8.2. Parameters**

None.

**2.9. Edit User**

**2.9.1. Description**

Edit the details of your own or another user.

If the invoking Role has not created any new users, or used [Create Reference To User](#) to make any references to existing users, the only user it will know about is its own. In this case, the task will bring up an edit screen for the details of the current user.

If, on the other hand, the invoking Role knows about more than one user, the task will bring up a screen showing a list of users, and fields in which user details can be entered. A user must be selected from the list, and details filled in - the entries will then be used to update the details of the selected user.

This task is placed by default in every new Role type. It can be deleted if not required.

**2.9.2. Parameters**

**User**

The user to be updated [shown only if the Role knows about other users than its own]

**Property name**

The nickname by which the user is to be known in the current Role

**Object name**

The user's name [optional - if not supplied, property name will be used as the default value]

**Email**

The user's email address

**Password**

The user's logon password

**Confirm Password**

The user's logon password

**Machine**

Is the user to be fully automated, i.e., not for human logon?

### **Offline**

Is the user's work to be carried out by a different system (a separate RADRunner installation, another process engine, or manually)?

## **2.10. External Component**

### **2.10.1. Description**

View a document, access a URL, or invoke an external function.

This task can be used in many different ways. The simplest is to view a file, which must be available via either http:// (the web) or file:// (on the LAN). Other uses include to view a web page (for instance, a search engine or company intranet) or access an application which has a web interface (for instance, a time/contact management or ERP system).

The more sophisticated uses of this task generally require parameters to be supplied to the base URL. The names and values of these parameters are taken from entity attribute values within the Role.

Note that there is no way to use this task to return values into the Role resources. If this is necessary, a different task must be used, such as [Access Relational Database](#) or [Web Service Call](#).

[How to add to a Role type](#)

### **2.10.2. Parameters**

None.

## **2.11. Get Entity From URL**

### **2.11.1. Description**

Update the Role with XML from an external source.

This task has some similarities to [External Component](#), in that it opens a URL for which both the location and the parameters are specified via entity attribute values. However, it is a non-interactive task which relies on the content of the URL being pure XML. If this is the case, the XML elements and attributes returned will be parsed into sub-entities and attributes in a specified entity within the Role.

This has various uses, for example:

## Process Enactment

- If the URL returns XHTML (HTML which conforms to XML syntax), this is a means of *web scraping*: retrieving data from an arbitrary web page and placing it into entity attributes within the Role. If the web page is dynamic this can be used to access data made available via the web.
- The task provides a means of calling functions or sending messages via the web without having to build web services which conform to the SOAP protocol. In particular, this could be used to access early web applications which provide XML over HTTP but which predate the SOAP specification.
- When calling SOAP services which are in-house, it can be simpler to use this technique than to make full-blown web service calls via the task [Web Service Call](#), in order to hide parameter complexity and allow Roles to supply only the key values. It is necessary to create a simple web page (for example, a JSP or ASP) to be invoked as the URL, which:
  1. Takes request parameters which indicate a service to call and the key parameter values to supply
  2. Invokes the corresponding service with default values for the remaining parameters
  3. Returns the XML response.

### [How to add to a Role type](#)

#### 2.11.2. Parameters

None.

## 2.12. Get Entity From XSL

### 2.12.1. Description

Update the Role with XML generated via XSL transformation of the Role resources.

This task has some similarities to [Get Entity From URL](#), in that it parses XML elements and attributes from an external source into sub-entities and attributes in a specified entity within the Role. However, in this case the XML is generated via XSL transformation of the Role resources.

To use the task, a stylesheet must be specified on definition, via a URL pointing to a \*.xsl file. The XSL in this file is applied to the resources of the Role in their [exported XML format](#). The resulting XML elements and attributes are parsed into sub-entities and attributes in a specified entity within the Role.

This has various uses, for example:

- To generate XML corresponding to a particular schema, for use as a web service call parameter

- To perform arbitrary text manipulation of entity attributes, to construct email message text
- To generate XHTML for use in a web page
- and so on.

To assist with the creation of stylesheets for role resource transformation, the task [Export Role Type Resources](#) can be used to export the resources of a particular role class as standard XML. This XML can then be loaded into a tool such as XMLSpy for use in defining an appropriate XSL transformation.

### [How to add to a Role type](#)

#### 2.12.2. Parameters

None.

## 2.13. Get Resource From Collection

### 2.13.1. Description

Extract an element from a list for processing.

Any entity within a Role can be regarded as owning 2 collections:

1. Its sub-entities
2. Its attributes.

Tasks carried out within a Role may result in these collections becoming, for a certain entity, lists of related items. For instance:

- A [Part Interaction Get](#) with *retain original* set to *true* will add the message to the receiving entity as a new sub-entity
- [Get Entity From URL](#) and [Get Entity From XSL](#) may result in an entity containing lists
- A [Web Service Call](#) may return lists in the response
- A [Put Resource Into Collection](#) may be used to add specific elements to a list.

In such cases, it can be useful to extract a particular element from the list for processing, viewing, editing, and so on. This task allows an index into such a list to be specified, via an entity attribute value. The list element corresponding to that index is then placed into a specified entity or attribute within the Role (if no element exists for the index, the task fails).

The collection used is determined by the type of the destination resource. If the destination resource is an entity, the sub-entities of the source entity are used as the collection. If the destination resource is an attribute, the attributes of the source entity are used as the collection.

## Process Enactment

If *retain original* is specified for the task, the element will be left in the collection. Otherwise it will be removed. This is one of 2 ways in which an element can be removed from a list: the other is if the element is sent via a [Part Interaction Give](#) with *retain original* set to *false*.

If *automated* is specified for the task, the selection is carried out in the background, i.e., based on the value of the index attribute, and without user involvement. Otherwise, the task page allows the element to be chosen from the collection, whose items are displayed on screen in a selection box (if no selection is made, the index attribute value will be used as in the automated case). *If the destination resource is an attribute*, the items in the source collection will also be attributes, in which case their values are displayed in the list. *If the destination resource is an entity*, the items in the source collection will also be entities, in which case the values shown are the values of a particular attribute in each item - the one named @, which corresponds to the *top-level text content of the item* when converted to [standard XML form](#). In either case, if no value exists, the index of the item will be shown enclosed in square brackets - for example, item 5 will be shown as [5].

### [How to add to a Role type](#)

#### 2.13.2. Parameters

None.

## 2.14. Introduce This Role Instance To Another Role Instance

### 2.14.1. Description

Set up an Interaction with an existing Role instance.

In some ways, this task is the inverse of [Start A Role Instance And Introduce It To This Role Instance](#). The differences are:

1. This task does not start a new Role as the interaction partner, it uses an existing Role instance. This must be specified when the task is added to a Role type; hence it must exist at process definition time, and be known to the Role which does the process definition.
2. The assumption is that many instances of the current Role type may be using this task to connect to the specified interaction partner. Hence, each time the task executes, new part interactions are created specially in the partner Role instance (whereas with [Start A Role Instance And Introduce It To This Role Instance](#), they are created in the invoker of the task).

A particular use of this task is to enable instances of [public Role types](#) to connect to existing processes. While any user can instantiate for themselves a public Role type known to the root Role, if they wish to interact with existing Roles it is necessary to make their new Role

known to others in the system - this task provides a means of doing so.

[How to add to a Role type](#)

#### **2.14.2. Parameters**

None.

### **2.15. Manual Action**

#### **2.15.1. Description**

Prompt the user to carry out a manual task.

Executing this task simply displays the description on screen, which reminds the user to carry out some manual action.

Since the task has no impact on Role resources, it cannot be used to record completion of the manual action. If this functionality is required, an alternative task should be used, such as [Presentation](#) or [Resource Attribute Maintenance Form](#).

[How to add to a Role type](#)

#### **2.15.2. Parameters**

None.

### **2.16. Part Interaction Get**

#### **2.16.1. Description**

Receive a message via an [interaction](#) with another Role.

Detailed explanation of interactions is covered under [Playwright semantics](#).

[How to add to a Role type](#)

#### **2.16.2. Parameters**

None.

### **2.17. Part Interaction Give**

## Process Enactment

### 2.17.1. Description

Send a message via an [interaction](#) with another Role.

Detailed explanation of interactions is covered under [Playwright semantics](#).

[How to add to a Role type](#)

### 2.17.2. Parameters

None.

## 2.18. Presentation

### 2.18.1. Description

An entry form or view screen of arbitrary complexity, created via XSL transformation of the Role resources.

This task provides a fully flexible means of displaying the resources of a Role instance, including to generate fully customizable maintenance forms for the resources.

The task is added to a Role instance specifying an attribute within that Role which contains the URL of a stylesheet. When the task is invoked, the stylesheet (if found) is used to transform the [standard XML form of the Role resources](#) (as provided by the Resources Snapshot window), and the resulting XHTML text is displayed to the user.

The displayed text could, for instance, include input elements: text areas, combo boxes, and so on. In this case, the user will be able to submit values back to the server as request parameters, with associated values (either default, or entered by the user). For each submitted parameter whose name corresponds to an attribute element in the compressed XML form of the Role resources, the corresponding Role resource attribute will be updated from the request parameter value.

For example, suppose a role *myRole* contains one resource *myInfo*, of the form:

```
<myRole>
  <myInfo>
    <name></name>
    <gender>Male</gender>
    <stylesheet>myHost/rim/in/myRolePresenter.xsl</stylesheet>
  </myInfo>
</myRole>
```

We could display an entry form containing an input text field for *name* and a drop-down

selection box for *gender* using the following stylesheet *myRolePresenter.xsl*:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" indent="yes" version="4.0"/>
  <xsl:template match="/">
    <table border="0">
      <xsl:for-each select="*/*">
        <xsl:element name="tr">
          <xsl:element name="td">
            <xsl:attribute name="width">50%</xsl:attribute>Name</xsl:element>
          <xsl:element name="td">
            <xsl:attribute name="width">50%</xsl:attribute>
            <xsl:element name="input">
              <xsl:attribute name="name">myInfo.name</xsl:attribute>
              <xsl:attribute name="value"><xsl:value-of select="name"/></xsl:element>
            </xsl:element>
          </xsl:element>
        </xsl:element>
      <xsl:element name="tr">
        <xsl:element name="td">
          <xsl:attribute name="width">50%</xsl:attribute>Gender</xsl:element>
        <xsl:element name="td">
          <xsl:attribute name="width">50%</xsl:attribute>
          <xsl:element name="select">
            <xsl:attribute name="name">myInfo.gender</xsl:attribute>
            <xsl:element name="option">
              <xsl:if test="gender='Male'"><xsl:attribute name="selected"
              Male
              </xsl:element>
            <xsl:element name="option">
              <xsl:if test="gender='Female'"><xsl:attribute name="selected"
              Female
              </xsl:element>
            <xsl:element name="option">
              <xsl:if test="gender='Other'"><xsl:attribute name="selected"
              Other
              </xsl:element>
            </xsl:element>
          </xsl:element>
        </xsl:element>
      </xsl:element>
    </xsl:for-each>
  </table>
</xsl:template>
</xsl:stylesheet>
```

This would result in a form looking like:

This is a simple case. Any text - HTML, XML, or anything else - can be created by this method, so the options open for presentation are literally infinite. For example, forms can be created which allow the user to enter certain information while showing other information as read-only. It is also possible to include Javascript in a form for dynamic behaviour - for

## Process Enactment

instance, to allow selections made in certain fields to update a hidden field which corresponds to a Role resource.

Note that the input to the XSL transformation - the [standard XML form of the Role resources](#) - has any spaces in entity/attribute names converted to underscores in order to avoid the occurrence of whitespace in XML element names. However, when processing values submitted via HTML input elements, RADRunner will attempt to match the "name" attribute of the input element against the true nested attribute name in the Role instance, which may contain whitespace. Hence, care should be taken when specifying the name attribute of input elements to use the Playwright name (with whitespace) if it differs from that shown in the standard XML form of the Role resources (with underscores).

To assist with the creation of stylesheets for role resource presentation, the task [Export Role Type Resources](#) can be used to export the resources of a particular role class as standard XML. This XML can then be loaded into a tool such as XMLSpy for use in defining an appropriate XSL transformation.

If all that is required is to make certain entity attribute values available to the user for viewing and/or editing, it may not be necessary to use this task. RADRunner also provides a simpler way of creating entry screens, via the task [Resource Attribute Maintenance Form](#).

[How to add to a Role type](#)

### 2.18.2. Parameters

As defined by the XHTML output of the XSL transformation.

## 2.19. Put Resource Into Collection

### 2.19.1. Description

Insert an element into a list.

See [Get Resource From Collection](#) for a discussion of collections inside a Role's resources. This task allows a collection element to be added or replaced.

The task allows an index into such a list to be specified, via an entity attribute value. A list element corresponding to that index is then updated from a specified entity or attribute within the Role:

- If the value is 0, the element is added at the start of the collection and all existing elements have their index incremented by 1
- If the value is between 1 and the current collection size, the existing element at that index

is replaced

- If the value is larger than the current collection size, the element is added at the end of the collection.

The collection used is determined by the type of the source resource. If the source resource is an entity, the sub-entities of the destination entity are used as the collection. If the source resource is an attribute, the attributes of the destination entity are used as the collection.

[How to add to a Role type](#)

### **2.19.2. Parameters**

None.

## **2.20. Resource Attribute Maintenance Form**

### **2.20.1. Description**

A simple entry form.

This task provides a simple means of allowing the user to view and edit entity attribute values within the Role. Each one is displayed in a field allowing input of multi-line text, along with a prompt to say what the field contains. The user can update whichever values they like. Validation of types (numeric, boolean, date) will be carried out on saving the changes, along with a check that the new values do not violate any [conditions](#) on the activity or in the Role generally.

The task does not allow for client-side validation of the content and format of values entered, or for customisation of the entry form. If this is required, a [Presentation task](#) should be used.

The task is set up at define time by specifying a number of entity attributes to be maintained, along with a corresponding set of entity attributes containing text prompts for each entry field.

[How to add to a Role type](#)

### **2.20.2. Parameters**

As defined by the entry form.

## **2.21. Send Email**

### **2.21.1. Description**

## Process Enactment

Send an email to your own or another user.

The task takes 3 parameters:

1. Recipient email address - taken from an entity attribute
2. Subject of the email - again, from an entity attribute
3. Body text of the email - from either an entity attribute or an entity. If the body is taken from an entity, the [standard XML format of the entity](#) is used as the text.

The sender email address is the email of the current user.

If it is desired to send email to another user (rather than, or as well as, using an interaction), the recipient user can be converted to an entity using the task [Convert User To Entity](#), and the recipient email taken from this entity.

It is also possible to send an email to yourself, for instance, to notify the user via email that something of significance has happened within one of their Roles. Leaving the recipient empty will cause the email to be sent to the email address of the current user.

For this task to work, the following system properties must be set:

- mail.host=*IP address of an SMTP mail server*
- user.name=*valid user of that SMTP mail server*

This can be done [in the RADRunner properties file or via the command line](#).

The SMTP server must also be visible to the server running RADRunner, and vice-versa, not hidden by a firewall - this can be tested using (for example) a PING command.

[How to add to a Role type](#)

### 2.21.2. Parameters

None.

## 2.22. Start A Role Instance And Introduce It To This Role Instance

### 2.22.1. Description

Start another Role and set up interactions with it.

This task is a fully automatable version of *Start Role Instance And Assign User*, with no parameters required. It also allows one interaction instance in each direction to be created and bound to the current Role instance and the new Role instance.

There are the following mandatory define time parameters, specified when the task is added

to a Role type:

- Role type to be instantiated
- Automated - whether or not the new Role instance should be automated

There are the following optional define time parameters, specified when the task is added to a Role type:

- User property name - entity attribute containing the property name of the new Role's user
- Interaction type for give
- Activity to add a new give part interaction task to - in current Role
- Get part interaction - in new Role
- Interaction type for get
- Give part interaction - in new Role
- Activity to add a new get part interaction task to - in current Role

There are the following optional runtime parameters, specified when the task is executed:

- User email
- User password
- User name

A user property name containing only whitespace is treated as null, i.e., unspecified.

If a user has been specified, and can be retrieved, it is used.

If a user has not been specified, an email has been specified, but no password has been specified, RADRunner looks for a user known to the Role with that email and uses it if found.

If a user is not found by either means:

1. If the task is automated, use the Role's own user, even if a user has been specified
2. If the part is manual, create a user (since in this case the new user's name will be displayed to the current user for information):
  1. If no user property name has been specified, generate a user property name
  2. If no email has been specified, set the email to the user property name
  3. If no password has been specified, set the password to the email
  4. If no name has been specified, set the name to the email

The entire algorithm is as follows:

1. Instantiate the specified Role type, automated as specified
2. Add a user if necessary, as above
3. Assign the Role instance to the user  
*Optionally, for each interaction type specified (there can be one in each direction):*
4. Instantiate it
5. Create a part interaction task in the current Role instance, in the specified activity, using

## Process Enactment

message name taken from the give/get in the interaction

6. Introduce the interaction instance to the current Role instance and the new Role instance created above.

### [How to add to a Role type](#)

#### 2.22.2. Parameters

**User email**

The email of the user of the new Role instance [optional]

**User password**

The password of the user of the new Role instance [optional]

**User name**

The name of the user of the new Role instance [optional]

## 2.23. Tell Role Instance About User

### 2.23.1. Description

Tell a Role instance that you have reference to, about a user that you have reference to - so that this Role instance also has the reference.

This can be used when, for example, another Role instance also needs to manage a process involving the user - for example, to delegate work to them.

If the invoking Role instance has not created any new users, or used [Create Reference To User](#) to make any references to existing users, the only user it will know about is its own.

This task is placed by default in every new Role type. It can be deleted if not required.

### 2.23.2. Parameters

**Role instance**

The role instance to be updated with knowledge of a new user

**Object to which a reference is held**

The user who will now be known to the specified Role instance

**Property name**

The nickname by which the user is to be known to the specified Role instance - if this nickname already exists in the specified Role instance, the task will fail

## 2.24. Tell Role Instance About Role Instance

### 2.24.1. Description

Tell a Role instance that you have reference to, about another Role instance that you have reference to - so that the first Role instance also has the reference.

This can be used when, for example, the first Role instance also needs to manage a process involving the second - for example, to reassign its user.

If the invoking Role instance has not created any new Role Instances, the only Role instance it will know about is itself.

This task is placed by default in every new Role type. It can be deleted if not required.

#### 2.24.2. Parameters

**Role instance**

The role instance (A) to be updated with knowledge of a new Role instance

**Object to which a reference is held**

The Role instance (B) who will now be known to Role instance A

**Property name**

The nickname by which Role instance B is to be known to Role instance A - if this nickname already exists in Role instance A, the task will fail

### 2.25. Tell Role Instance About Interaction Instance

#### 2.25.1. Description

Tell a Role instance that you have reference to, about an interaction instance that you have reference to - so that this Role instance also has the reference.

This can be used when, for example, another Role instance also needs to manage a process involving the interaction - for example, to change the Role instances bound to it.

This task is placed by default in every new Role type. It can be deleted if not required.

#### 2.25.2. Parameters

**Role instance**

The role instance to be updated with knowledge of a new interaction instance

**Object to which a reference is held**

The interaction instance that will now be known to the specified Role instance

**Property name**

The nickname by which the interaction instance is to be known to the specified Role instance - if this nickname already exists in the specified Role instance, the task will fail

## 2.26. Upload File

### 2.26.1. Description

Upload a file to the server, storing a reference to the uploaded file path in the Role resources

This task prompts the user to specify a file on their own LAN, which will then be uploaded to a specified filename on the server. Both prompt and filename are taken from resource attributes in the Role instance, specified at define time. The filename should have no path specified - it is placed in the [in directory](#), with a prefix of the role instance id in order to distinguish it from files uploaded via other role instances.

A URL to the uploaded file is also placed in a third resource attribute, also specified at define time. If this resource attribute is used as the target for an [External Component](#) task, the uploaded file can then be viewed and/or edited from within the role instance. If the attribute was sent to another role instance via an interaction, that role instance could also view and/or edit the file in the same way.

The task [External Component](#) can be used to view uploaded files from within a web browser.

In conjunction with [Download File](#), this task can be used to implement *web-based document management* via RADRunner: structured file transfer and maintenance by multiple users.

[How to add to a Role type](#)

### 2.26.2. Parameters

None.

## 2.27. Web Service Call

### 2.27.1. Description

Invoke a web service.

This task calls a web service operation, using as parameters either constants or values taken from the Role resources. If an entity resource rather than an entity attribute is specified, the [standard XML form of the entity](#) will be used.

Return values can be placed into entity attributes or entities within the Role. This is optional, and unused return values will be described in the RADRunner status messages but otherwise ignored. If an entity rather than an entity attribute is specified, the return value is assumed to

be in [standard XML form](#) and parsed accordingly into attributes and sub-entities.

RADRunner provides a powerful facility for web service invocation, in that parameters of complex type can be specified without using programming tools to generate "stub classes". To do this, use an entity as parameter whose [standard XML form](#) matches that of the complex type concerned. In particular, the entity should contain a top level attribute named *@xsi:type* with value corresponding to the XML type of the parameter.

[How to add to a Role type](#)

### **2.27.2. Parameters**

None.