# Raduga

## *Developer Guide*

Raduga 1.07.0013.003

## Table of Contents

# General information

## Copyright



## Contacts

For any questions and support regarding this product, contact Michael Dvorkin (tel +79185402272, support@LazyDeploy.com).

## Licensing

Raduga Free software can be used for free. It is restricted to 5 environments and 50 projects. Free edition has a limited technical support.

Raduga Pro software can be used for free during the trial period of 30 days. After the end of the trial period, you must install a private license for each user to continue using the software. Raduga Pro can manage an unlimited number of environments and projects and it has full technical support.

Contact Michael Dvorkin (tel +79185402272, support@LazyDeploy.com) to obtain Raduga licenses.

## Disclaimer

Raduga allows deleting database and file system objects. In some cases the objects are replaced during the migration of development projects. Raduga users should carefully test all development projects in a test environment before implementing them in production. We accept no liability for any damage caused by the Raduga application. Object transmission cannot be guaranteed to be secure or error-free, as migration rules can differ from one environment to other. We therefore do not accept liability for any errors or omissions in the contents of custom objects which might arise as a result of object transmission. Although we have taken reasonable precautions to ensure proper performance of Raduga software, the company cannot accept responsibility for any loss or damage arising from the use of Raduga.

# About Raduga

Raduga is an application that helps you manage the development and deployment process. It is designed for Oracle applications; however, it can be used in any development environment. A user-friendly interface, easy navigation between applications and projects, various migration and deployment capabilities, version control and reporting make Raduga a useful tool for programmers, team leaders and project managers.

Raduga offers to users

- Object migration between environments
- Intuitive navigation between entities
- Object comparison
- Version control and deployment history
- Monitoring environment status
- Starting/stopping environments
- Data loading capabilities
- Easy customization
- Comprehensive reporting
- File transfer capabilities
- Enhanced security

# Raduga Development Concepts

## Internal Architecture

Raduga stores the complete definition and architecture of the target environment, as well as a set of rules for working with the database and file system objects in the target environment. To do this, Raduga defines several metadata objects (services, entities, entity types, steps and constants) that fully describe the database, file system, E-Business Suite and other environments.

Raduga's configuration definition is stored in XML files in the Raduga global configuration directory. The path to Raduga global configuration directory can be found in the HKLM\Software\Raduga6\ConfigDir registry key. All XML files of the form Raduga_*.xml in the Raduga global configuration directory are treated as Raduga configuration files.

Raduga installs eight configuration files by default:

- Raduga_UTIL.xml Stores general definitions used by other configuration files
- Raduga_DB.xml Defines Raduga utilities for database objects
- Raduga_EBS.xml Defines Raduga utilities for E-Business Suite objects
- Raduga_FILE.xml Defines Raduga utilities for files
- Raduga_APEX.xml Defines Raduga utilities for Oracle Apex
- Raduga_SETUP.xml Defines Raduga utilities for E-Business Suite setup
- Raduga_CLOUD.xml Defines Raduga utilities for Oracle Cloud
- Raduga_Custom.xml A custom configuration file that stores all custom definitions as well as Raduga seeded utilities that have been modified

All Raduga objects have a prefix that depends on the configuration file that defines the object. For example, all objects defined in the Raduga_UTIL.xml file have the “util” prefix (util.find_files, util.SETUP_DIR). All objects defined in Raduga_EBS.xml have the “ebs” prefix (ebs.apache_start).

Custom objects may have a prefix. We strongly recommended that you follow Raduga’s development standards and create custom objects with custom prefixes. The default custom prefix is “xxx”. It is defined in the “util.CUSTOM_PREFIX” variable and can be changed to a preference of your choice by a Raduga Administrator.

The following sections describe Raduga objects used for defining Raduga environments. They are:

- Services
- Entities
- Entity Types
- Steps

- Constants

## Services

Modern environments usually have a distributed architecture and may be installed on multiple servers. Each server has its own specialization. For example, the E-Business Suite system may consist of a database, concurrent, forms, and web servers. To describe the complex architecture, Raduga uses a special object called a “service”. Currently there are several built-in services defined in Raduga:

- db.database – a database service hosted by the database server
- db.listener – a database listener service hosted by the database server
- ebs.concurrent – the concurrent manager server hosts this service
- ebs.mailer – a mailer service hosted by the concurrent manager server
- ebs.discoverer – a discoverer service hosted by the discoverer server
- ebs.discoverer_infra – a discoverer infrastructure service hosted by the discoverer server
- ebs.forms – a forms service hosted by the forms server
- ebs.framework – a framework service hosted by the web server
- ebs.opmn – an opmn service hosted by the web and forms server
- ebs.reports – a service hosted by the concurrent manager server (obsolete in R12)
- ebs.web – a web service hosted by the web server
- util.file – the files server as well as all other server types can host this service

Raduga allows adding custom services to satisfy organization needs. Every server defined in the system can host one or more services.

## Entities

An entity defines rules for working with the database or file system objects. Here are some examples of entities defined in Raduga:

db.Packages

db.Users

ebs.Alerts

ebs.Forms

Currently there are about 70 built-in entities in Raduga. You can create a custom entity for the objects that are used in your client system. Entities are visible to Raduga users.

Every entity is based on an internal entity type that defines basic operations that can be performed on the entity object. Every entity has a list of services that it works on. According to this list Raduga decides how to deploy objects belonging to a specific entity.

## Entity Types

Every entity has its own entity type that defines the low level technical operations that can be performed with the entity object. For example, the “db.package” entity type contains database commands that are executed in order to list all database packages, save them to file and compile them.

## Steps

Steps are procedures executed on the server. Examples of procedures are “Start Apache Server” or “Compile APPS schema”. Raduga decides which server to execute a step on according to the services defined for the step and hosted by the server.

## Constants

Constants define the behavior of entities and steps. Raduga defines several built-in constants:

- ebs.UPDATE_WHO_FIELDS defines whether “created/update by” and “create/update date” fields are updated during object migration
- db.UPDATE_DB_OBJECTS defines whether Raduga allows users to delete or rename database objects
- ebs.UPDATE_EBS_OBJECTS defines whether Raduga allows users to delete or rename E-Business Suite objects
- util.UPDATE_FS_OBJECTS defines whether Raduga allows users to delete or rename files

# Defining Entity Types

## General Concepts

An Entity type is a central configurable Raduga class that defines migration rules. In general, for every object that can be migrated by Raduga it is necessary to define at least three operations:

- List – how to get a list of all objects
- Get – how to get the object from the server
- Put – how to put the object on the server

Additionally you can define the following operations:

- Deploy – how to install the object after it was sent to the server
- Delete – how to delete an object from the server
- Rename – how to rename the object on the server

## Operation "List"

To list all database packages, their last compile time and their status, run the following query:

```
select object_name, last_ddl_time, status
  from user_objects
 where object_type = 'PACKAGE BODY'
```

You can use this statement to define the "List" operation for the database package entity type.

In fact the operation should be written as a shell script using "sh" syntax according to special rules that Raduga uses. Here is the "list" operation for the "db.package" entity type, from the Raduga_EBS.xml configuration file:

```
Step{util.define_env}

'sqlplus' -s ${DBCREDENTIALS} << SQL
set pages 0
set lines 32767
set trimspool on
set feed off
set wrap on
select trim(object_name)||'|'||to_char(last_ddl_time,'DD/MM/YYYY HH24:MI')||'|'||status
  from user_objects
 where object_type = 'PACKAGE BODY'
   and upper(object_name) like upper(replace('${MASK}','*','%'))
 order by object_name;
exit;
SQL
Step{util.Exit}
```

The script uses some built-in variables:

**${DBCREDENTIALS}** variable stores database user/password

**${MASK}** variable stores a UNIX type wildcard entered by user in order to restrict the search to only relevant objects

Note that Raduga variables should be enclosed in {} brackets in a script.

**Step{uti.define_env}** is a sub procedure defined in the Raduga_UTIL.xml configuration file. The file contains technical sub procedures that can be used in Raduga scripts. Raduga replaces the Step{util.define_env} string with the actual code that defines the environment for Raduga scripts. The util.define_env procedure contains commands for defining a script's environment.

**Step{uti.Exit}** is a sub procedure defined in the Raduga_UTIL.xml configuration file. The util.Exit procedure contains commands for correct exiting from the shell script.

All calls to standard UNIX or Linux commands should be enclosed in apostrophes (for example, 'sqlplus') in order to avoid using custom aliases.

Use at least the following sqlplus formatting instructions to create output that Raduga can parse:

```
set pages 0
set lines 32767
set trimspool on
set feed off
set wrap on
```

Dates should have the following format: DD/MM/YYYY HH24:MI

SQL query output should be a list of fields separated by pipes (the "|" sign). You can achieve this in two ways:

- By concatenating the columns

  select trim(column1) || '|' || trim(column1) from …

- By setting the column separator to "|"

  set colsep '|'

Here is another example of listing ADI integrators:

```
Step{util.define_env}

'sqlplus' -s ${DBCREDENTIALS} << SQL
set pages 0
set lines 32767
set feed off
set wrap off
```

```
set def off
set colsep '|'
col user_name format a100
col layout_code format a100
col ddl_time format a22
select t.user_name, t.integrator_code, to_char(greatest(t.last_update_date, t.creation_date),'DD/MM/YYYY HH24:MI')
ddl_time
  from bne_integrators_tl t,
       fnd_application a
 where t.application_id = a.application_id
   and a.application_short_name = upper('${APP}')
   and t.language = '${LANGUAGE}'
   and upper(t.user_name) like upper(replace('${MASK}','*','%'));
exit;
SQL
Step{util.Exit}
```

In this example there are two additional built-in variables:

**${APP}** stores the application code chosen by the user (for example, SQLGL)

**${LANGUAGE}** stores the language code (for example, US)

## Operation "Get"

For the objects existing on the server there is no need to implement a "get" operation. However, it is necessary to download database objects from the database as a part of a "get" procedure. Here is an example of downloading the ADI Integrator ("get" operation for the "ebs.adi_integrator" entity type):

```
Step{util.define_env}
Step{ebs.get_nls_lang}

if [ $STATUS -eq 0 ]
then
   FNDLOAD ${DBCREDENTIALS} O Y DOWNLOAD $BNE_TOP/patch/115/import/bneintegrator.lct
"${STAGE}/${OBJECT_HASH}" BNE_INTEGRATORS INTEGRATOR_ASN="${APP}" INTEGRATOR_CODE="${OBJECT_2}"
1>${ERR_FILE} 2>&1
          logfile=`'cat' ${ERR_FILE} | 'grep' '.log' | 'awk' '{print $NF}'`
          if [ ! -f "$logfile" ]
          then
            MESSAGE=`'cat' ${ERR_FILE}`
            Step{util.Failure}
          else
            Step{ebs.check_fnd_status}
          fi

   Step{util.rcs_co}
fi
Step{util.Exit}
```

Here is an explanation of some commands from the example:

**Step{util.define_env}** defines the environment for the current shell.

**Step{ebs.get_nls_lang}** defines the NLS_LANG environment variable according to the language chosen by the user. It is required so the FNDLOAD utility can work correctly.

**$STATUS** variable is defined in **Step{util.define_env}** and can be used in all Raduga procedures. Note that the $STATUS variable is a shell script variable, so it does not have to be enclosed in brackets {}. Only Raduga's built-in variables should always be enclosed in brackets {}.

**FNDLOAD** is a standard Oracle utility used for downloading/uploading Oracle Applications database entities.

**${STAGE}** is a Raduga built-in variable that defines a path on the server where Raduga can store its temporary and RCS objects.

**${OBJECT_HASH}** is a Raduga built-in variable that stores the unique hash value of the current Raduga object. Raduga uses hash strings instead of real object names because object names can contain spaces as well as national language characters that cannot be used in file names.

**${OBJECT_2}** is a Raduga built-in variable that contains a value from the second column of the object list (in this example, an integrator code).

**${ERR_FILE}** is a Raduga built-in variable that contains a name of the error file for the current session.

**Step{util.Failure}** is a standard Raduga utility that is called if the process fails.

**Step{ebs.check_fnd_status}** is a standard Raduga routine that is able to parse the $logfile and decide whether the FNDLOAD utility was successful.

**Step{util.rcs_co}** is a standard Raduga utility that is responsible for checking out the object from the Raduga RCS (Revision Control System).

**Step{uti.Exit}** is a sub procedure that contains commands for correct exiting from the shell script.

Every operation should be finished by the **Step{uti.Exit}** command to return to Raduga the current operation status.

## Operation "Put"

For existing file system objects it is not necessary to define "put" operation. For database objects it is necessary to define rules for uploading the objects to the database.

Here is an example of a "put" operation for the "ebs.adi_integrator" entity type:

```
Step{util.define_env}
Step{util.rcs_ci}

if [ ${STATUS:-0} -eq 0 ]
then
```

```
   Step{ebs.get_nls_lang}
   Step{ebs.update_who_fields}
   FNDLOAD ${DBCREDENTIALS} O Y UPLOAD $BNE_TOP/patch/115/import/bneintegrator.lct
"${STAGE}/${OBJECT_HASH}" - WARNING=YES ${UPLOAD_MODE:-} CUSTOM_MODE=FORCE 1>${ERR_FILE} 2>&1
          logfile=`'cat' ${ERR_FILE} | 'grep' '.log' | 'awk' '{print $NF}'`
          if [ ! -f "$logfile" ]
          then
            MESSAGE=`'cat' ${ERR_FILE}`
            Step{util.Failure}
          else
            Step{ebs.check_fnd_status}
          fi
fi

if [ -f "${STAGE}/${OBJECT_HASH}" ]
then
  'rm' -f ${STAGE}/${OBJECT_HASH}
fi

Step{util.Exit}
```

Here is the explanation of some commands from the example:

**Step{util.rcs_ci}** checks the object into the Raduga RCS repository.

**Step{ebs.update_who_fields}** updates the "who" fields (update date and updated by) in the "LDT" file that contains object definitions for the FNDLOAD utility.

**${UPLOAD_MODE}** is defined in the Step{util.define_env} procedure and defaults to "UPLOAD_MODE=REPLACE". However it can be overridden by the Step{ebs.get_nls_lang} utility and set to "UPLOAD_MODE=NLS" when an NLS object is uploaded.

**Step{ebs.check_fnd_status}** is a standard Raduga routine that can parse the $logfile and decide whether the FNDLOAD utility was successful.

## Operations "Delete" and "Rename"

To delete or rename an object from the server, Raduga uses a standard Oracle API or direct database update. However, for some objects the "Delete" operation is not possible because Oracle Applications does not supply an API or direct database operation is not allowed. Many organizations do not allow deleting and renaming Oracle Applications objects in any way other than via Oracle Applications, so by default Raduga does not allow deleting/renaming objects. However, the Raduga administrator can change this behavior by updating these Raduga built-in constants:

${ebs.UPDATE_EBS_OBJECTS} = Y – to allow deleting and renaming application objects (default: N)
${ebs.UPDATE_DB_OBJECTS} = Y – to allow deleting and renaming database objects (default: N)
${ebs.UPDATE_FS_OBJECTS} = Y – to allow deleting and renaming file system objects (default: N)

Here is an example of "delete" operation for the "ebs.adi_integrator" entity type:

```
Step{util.define_env}
```

```
if [ "${ebs.UPDATE_EBS_OBJECTS}" == "Y" ]
then

'sqlplus' -s ${DBCREDENTIALS} << SQL 1>${ERR_FILE} 2>&1
set def off
declare
   m_app_id fnd_application.application_id%type;
          m_res number := 0;
begin
   select application_id
            into m_app_id
            from fnd_application
           where application_short_name = '${APP}';

   m_res := BNE_INTEGRATOR_UTILS.DELETE_INTEGRATOR (p_application_id  => m_app_id,
                                      p_integrator_code => '${OBJECT_2}');
   commit;
end;
/
exit;
SQL
   sts=$?
          res=`'cat' ${ERR_FILE} | 'egrep' 'ORA-|PLS-'`
   if [ $sts -ne 0 -o -n "$res" ]
   then
     MESSAGE=`'cat' ${ERR_FILE}`
     Step{util.Failure}
   fi

else
  MESSAGE="Raduga administrator has disabled this action"
  Step{util.Failure}
fi

Step{util.Exit}
```

Here is an example of the “rename” operation for the “ebs.adi_integrator” entity type:

```
Step{util.define_env}

if [ "${ebs.UPDATE_EBS_OBJECTS}" == "Y" ]
then

'sqlplus' -s ${DBCREDENTIALS} << SQL 1>${ERR_FILE} 2>&1
set def off
update bne_integrators_tl t
  set t.user_name = '${NEW_OBJECT}'
```

```
 where t.integrator_code = '${OBJECT_2}'
   and t.language = '${LANGUAGE}';

commit;
exit;
SQL

res=`'cat' ${ERR_FILE} | 'grep' ^ORA-`
if [ -n "$res" ]
then
   MESSAGE=$res
   Step{util.Failure}
fi

else
   MESSAGE="Raduga administrator has disabled this action"
   Step{util.Failure}
fi

Step{util.Exit}
```

Here is the explanation of some commands and variables from the example:

**Step{util.define_env}** defines environment for current shell.

**${DBCREDENTIALS}** stores the database user/password.

**${ERR_FILE}** contains the name of the error file for the current session.

**${NEW_OBJECT}** contains the new object name entered by the user during the “rename” operation.

**${APP}** stores the application code chosen by the user (for example, SQLGL).

**${LANGUAGE}** stores the language code (for example, US).

**$STATUS** this variable is defined in **Step{util.define_env}** and can be used in all Raduga procedures.

**${OBJECT_2}** contains a value from the second column of the object list (in this case it’s an integrator code).

**Step{util.Failure}** a standard Raduga utility that is called if the process fails.

**Step{ebs.check_fnd_status}** parses the $logfile and determine whether the FNDLOAD utility was successful.

**Step{util.rcs_co}** checks out the object from the Raduga RCS (Revision Control System).

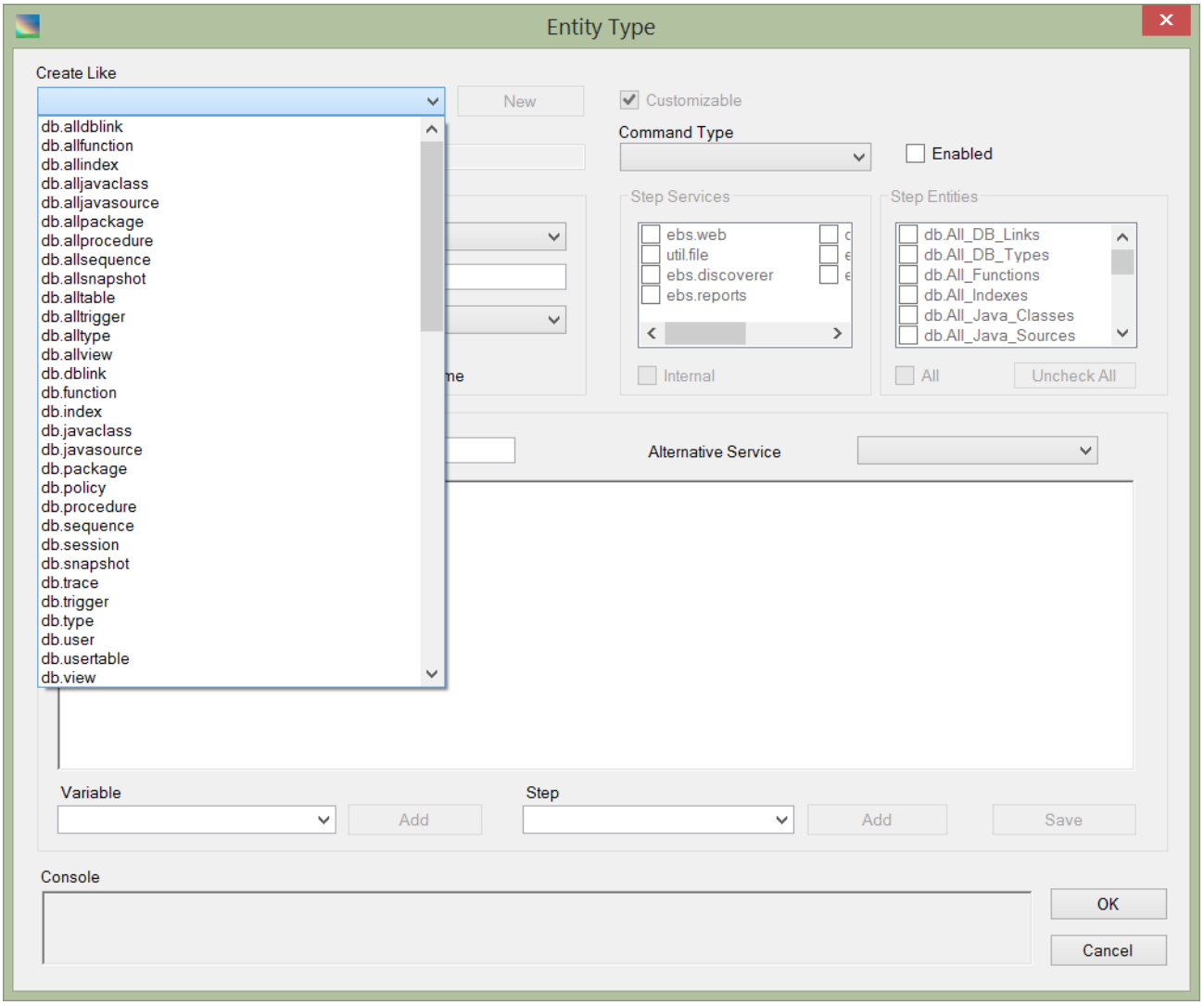
### Operation “Deploy”

For some objects it is necessary to perform additional commands during their deployment. These commands can be added as a part of the “deploy” operation.

## Entity Types customization

Entity types cannot be changed. However, you can alter entity type behavior by customizing the steps that comprise it. See Appendix C for the list of Raduga built-in customizable steps.

You can also create your own custom entity types to archive a behavior specific to your company's needs. To do that, open the "New Entity" form by going to "Admin" -> "Global Configuration". Select "Entity Types" in the "Objects" drop down and click "Edit". In the entity types list form click "Add". In the "New Entity" form choose an existing entity type in the "Create As" drop down:




This will bring the full definition of the selected entity type and replace its prefix by the custom prefix defined by the "util.CUSTOM_PREFIX" constant. You can edit the new entity type and modify the existing code, or add your own code to its definition.

## Creating User Values

In some cases Raduga needs to get additional input from the end user while transferring an object to or from the server. For example, when you import the discoverer workbook you need to know information like the workbook owner and responsibility. To get this information from the end user add the following code to the “put” operation of the ebs.discoverer_report entity type:

```
<RadugaValues>
        <condition value1="x" value2="x" />
        <variable name="MYUSER" type="String" title="Workbook Owner" default="${OBJECT_3}" sql="select
user_name from fnd_user" />
        <variable name="MYPASSWORD" type="Password" title="Workbook Owner Password" default="" sql="" />
        <variable name="MYAPPSMODE" type="String" title="Apps Mode" default="Y" sql="Y,N" />
        <variable name="MYAPPSRESP" type="String" title="Responsibility" default="" sql="select
resp.responsibility_name from fnd_user_resp_groups urg, fnd_responsibility_vl resp where urg.responsibility_id =
resp.responsibility_id and urg.responsibility_application_id = resp.application_id and urg.user_id = (select user_id from
fnd_user where user_name = upper('${PARAM0}'))" />
        <variable name="MYEUL_US" type="String" title="EUL_US" default="${EULUSER}" sql="" />
        <variable name="MYADM" type="String" title="Add admin privilege to Workbook Owner?" default="Y" sql="Y,N"
/>
</RadugaValues>
```

Code explanation:

**<RadugaValues>** XML tag opening/closing the code

**<condition>** The code will be executed only if value1 is equal to value2. In the example, the code will always execute. However there are cases when the code should be executed only for specific files only. Look at the example:

```
<condition value1="${REMOTEEXTENTION}" value2="class" />
```

In this case the code will only execute for files with a .class extension.

**<variable>** Variable definition. It consists of the following parts:

**name** variable name. It can be used further in the entity operation code. Here is an example:

```
eulapi -connect ${MYUSER}/${MYPASSWORD}@${ENV}
```

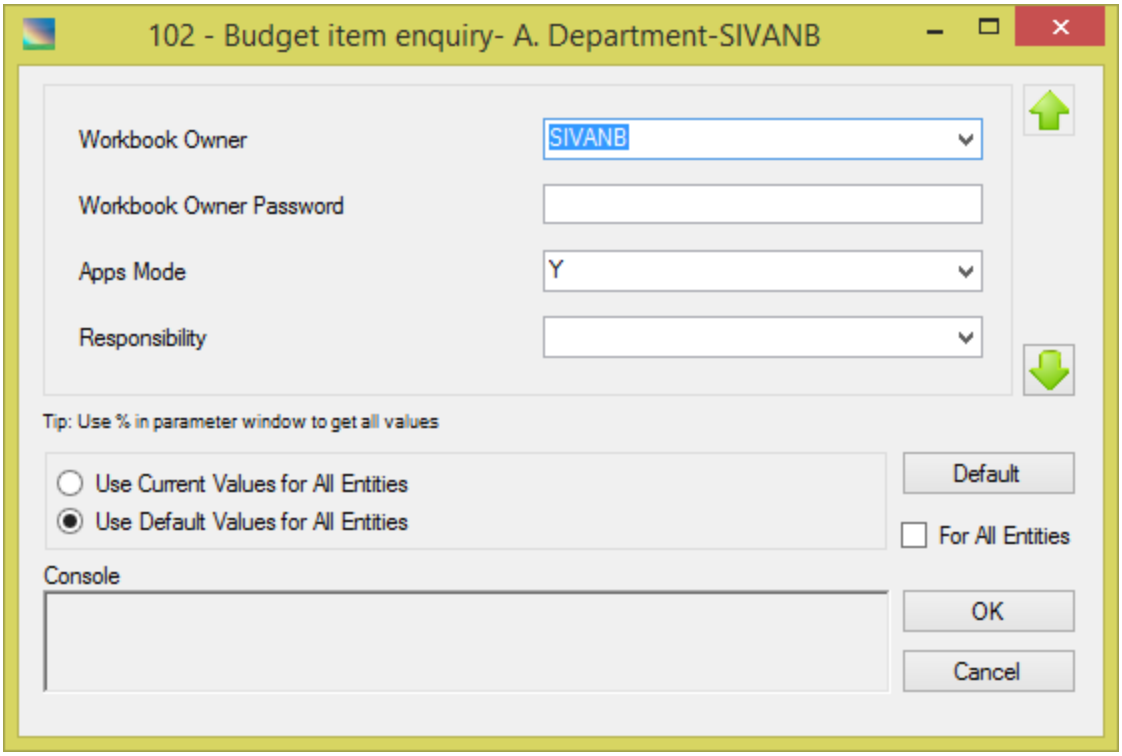
**type** variable type. Valid values: String, Int, Password, Date

**title** variable display name

**default** variable default value

**sql** variable valid values (see section “Defining Valid Values”)

This code will cause the user values form to appear during discoverer report deployment:




The end user enters values for all the required parameters. If there are more than four parameters (which is true in our case), use the up and down arrows to navigate through the list. The parameters with “sql” populated in the code are displayed as drop down valid values lists. Some of the parameters have default values.

In the form:

| | |
|---|---|
| **Workbook Owner** | A title taken from the “title” node of the previous code |
| **Use Current Values for All Entities** | Select to display currently-entered values for all discoverer reports. |
| **Use Default Values for All Entities** | Select to display default values for all discoverer reports. |
| **Default** | Press to restore default values for all parameters. |
| **For All Entities** | Select to accept the current or default values, and not display the form again. |

## Defining Valid Values

In some cases when defining parameter you need to construct a valid values list for the parameter. Several commands that can create the parameter's valid values list:

- **List**

  Comma separated list of valid values. Example: Y, N

- **One of the built-in Raduga variables:**

  [ENVIRONMENTS] – all existing Raduga environments in long name format (like EBS.TST, DB.PROD)

  [SHORTENVIRONMENTS] – all existing Raduga environments in short name format (like TST, PROD)

  [ENTITIES] – all existing Raduga entities

  [APPLICATIONS] – all applications existing in the environment

  [LANGUAGES] – all languages existing in the environment

  [STATUSES] – all Raduga statuses

- **SQL statement**

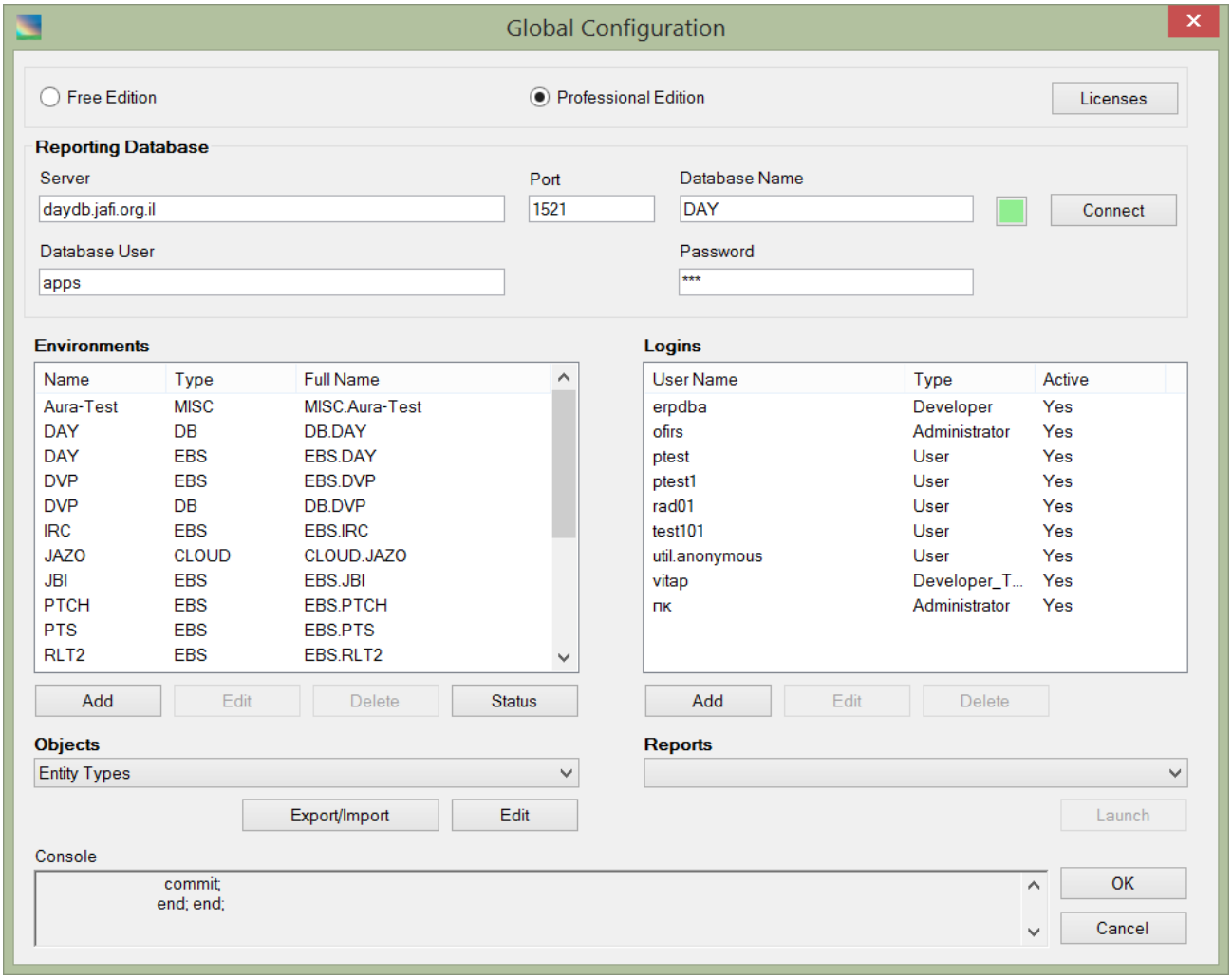
  A valid SQL statement that returns one column. In the “where” part of the SQL statement it is possible to use a value of other parameters: ${PARAM0}, ${PARAM1}, ${PARAM2} etc.

## Adding/Updating Entity Types

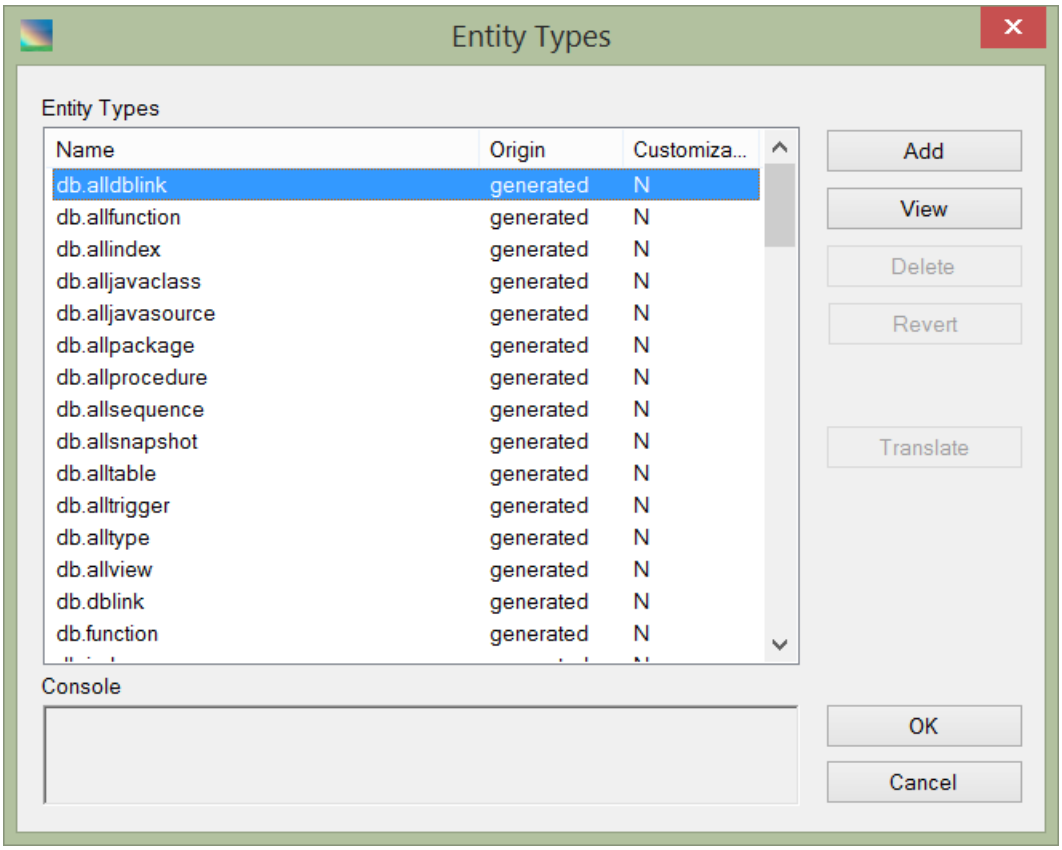
Raduga lets you edit some of the existing entity types and add new ones. All changes to existing "seeded" entity types and all new entity types are saved in the Raduga_Custom.xml configuration file. Only the Raduga administrator can add or edit entity types.

To add or edit entity types:

1. Press "Admin" on the Raduga main form.
2. In the "Private Configuration" window that appears, press "Global Configuration".
3. In the "Global Configuration" window choose "Entity Types" in the "Objects" drop down menu and press "Edit":

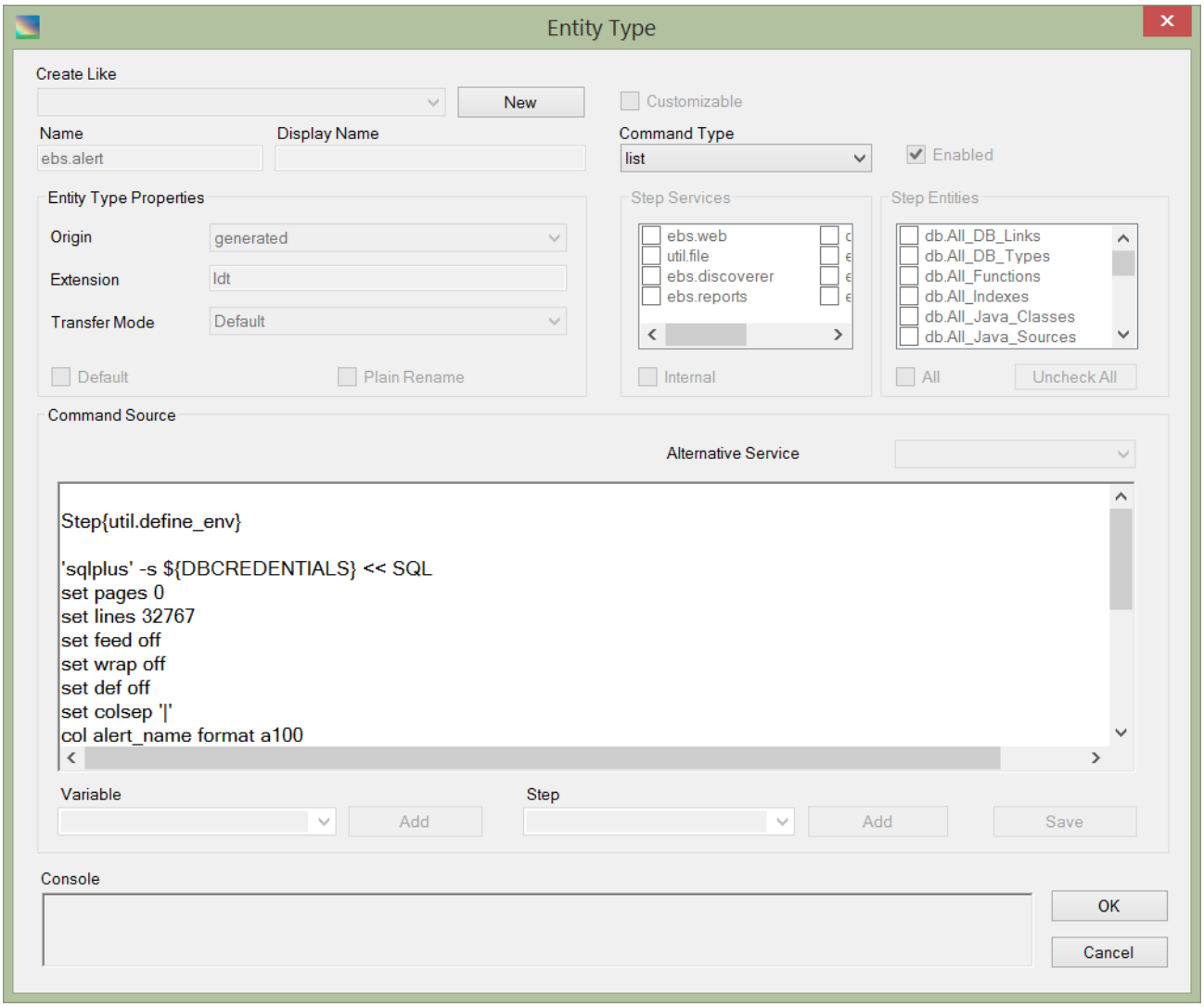


The “Entity Types” window appears:




- To view/edit an existing entity type, select the entity type in the “Entity Types” list box and click “View/Edit”. For customizable entity types the “Edit” button is enabled. For the read-only default entity types a “View” button is available instead of the “Edit” button.
- To add a new entity type, press “Add”.

The “Command” window appears:




The following fields and controls are available:

**Create Like** Choose an entity type from the drop down list to use as a template for the new entity type.

**New** Press to create a new entity type.

**Customizable** A read-only box that indicates whether the entity type can be edited. If the “Customizable” box is unchecked the entity type can only be viewed.

**Name** Entity Type name.

**Command Type** Choose an operation and provide a shell script for it in the “Command Source” box.

**Enabled** Select to enable the entity type.

**Origin** The entity type origin has two values: “existing” for files and “generated” for database objects (or any objects that are not files).

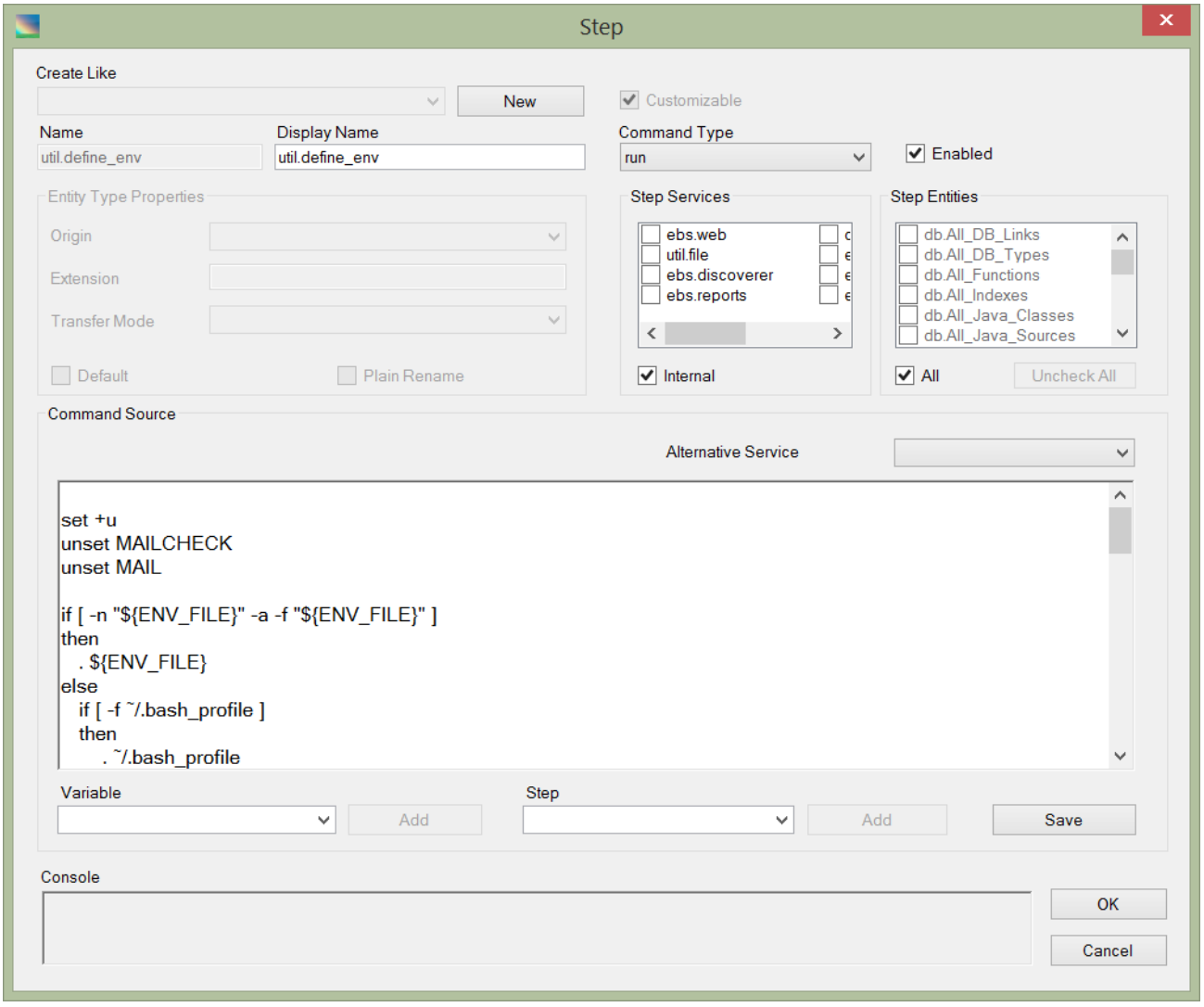
| | |
|---|---|
| **Extension** | Downloaded file extension (for example, “ldt” for files downloaded by the FNDLOAD utility). Can be empty for existing entity types. |
| **Default** | Select to use this as the default entity type for all entities. |
| **Plain Rename** | Select to allow objects of this type to be renamed in a single step. Deselect to require the object to be deleted and re-created when you rename it. |
| **Command Source** | Enter the shell script that implements the current operation (command type). |
| **Alternative Service** | Add an alternative service for the current operation (usually used for external applications like Discoverer). |
| **Variable** | Choose a built-in variable to add to the shell script. |
| **Step** | Choose a built-in step to add to the shell script. |
| **Save** | Save the current command source code. |

## Defining Steps

A step is a Raduga built-in procedure that executes an auxiliary operation. Example of such operation can be “Start Concurrent Server” or “Compile JSP”. Raduga lets you edit some of the existing steps and add new ones. Raduga saves all changes to existing, “seeded” steps and all new steps in the Raduga_Custom.xml configuration file. Only the Raduga administrator can add or edit steps.

To add or edit steps:

1. Press “Admin” on the Raduga main form.
2. In the “Private Configuration” window press “Global Configuration”.
3. In the “Global Configuration” window choose “Steps” in the “Objects” drop down menu and press “Edit” to display the “Steps” window.




In the “Step Services” region check all services that are relevant to the current step. For example, the step “ebs.cm_start” starts the concurrent manager, so the relevant service is “ebs.concurrent”.

In the “Step Entities” region select “All” or check all entities that are relevant to the current step. For example, the step “ebs.cm_start” is a general step that does not depend on entity, so “All” should be checked.

The “Compile Form” step compiles server form, so the relevant entity is “ebs.Forms”.

Check the “Internal” check box if the step is used for internal purposes and shouldn’t be shown to the end user.

The “Display Name” may be left blank for internal steps. Steps shown to the end user should have a display name.

Raduga lets you customize some of the seeded steps to change the behavior of Raduga entities. There are several customizable steps:

**util.define_env** is a general procedure that is executed at the beginning of each operation. It contains general shell script environment definitions. Here is its code:

```
set +u
unset MAILCHECK
unset MAIL

if [ -n "${ENV_FILE}" -a -f "${ENV_FILE}" ]
then
  . ${ENV_FILE}
else
  if [ -f ~/.bash_profile ]
  then
     . ~/.bash_profile
  else
     if [ -f ~/.profile ]
     then
       . ~/.profile
     fi
  fi
fi

set +u
unset MAILCHECK
unset MAIL

UPLOAD_MODE="UPLOAD_MODE=REPLACE"; export UPLOAD_MODE
STATUS=0; export STATUS
MESSAGE=""; export MESSAGE

# --------------
# Custom actions
# --------------
case "${ENTITY}" in

# You can add your code here
# This custom code will be executed for each entity:

# Example:
```

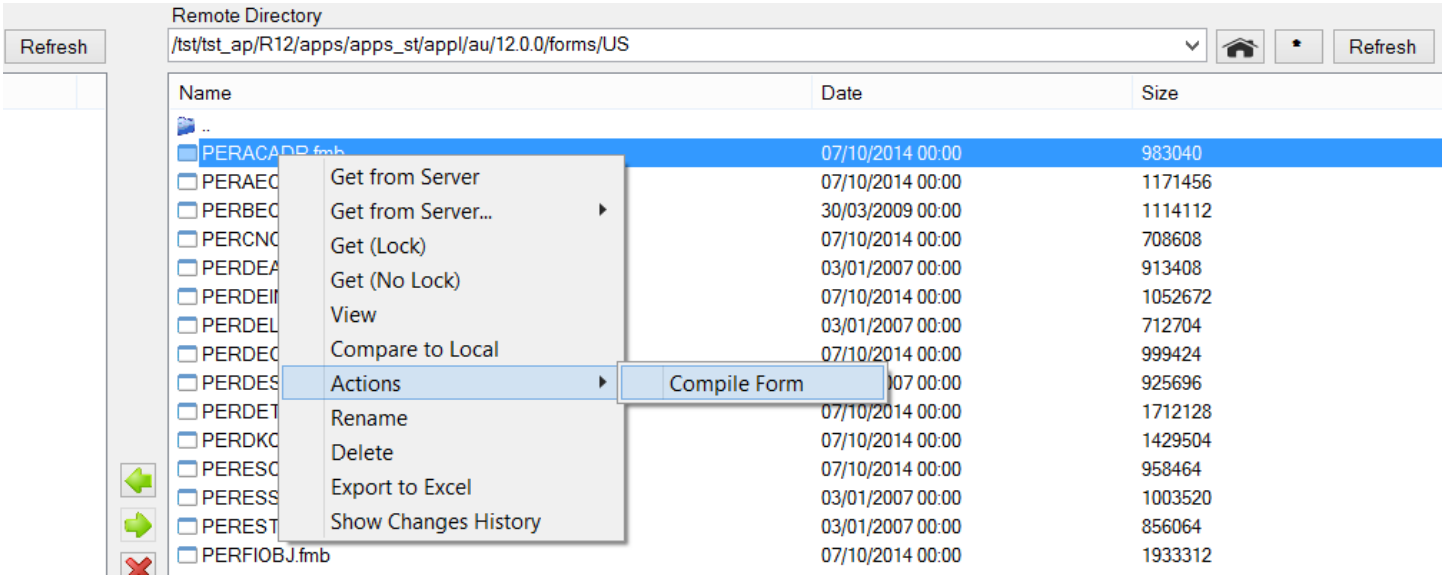
```
# ebs.Form_Functions)
# Your code here
#  ;;
# ebs.Programs)
# Your code here
#  ;;
# ebs.Forms)
# Your code here
#  ;;

*)
  ;;

esac

# This custom code will be executed for all entities:
# Your code here

# ---------------------
# End of custom actions
# ---------------------
```

The code contains a special place for adding custom commands for changing script behavior. The custom code can be added inside the “case” statement if it is designed for a specific entity or outside the “case” statement if it is general code.

Another step that can be customized is **util.Exit**. This step is called at the end of each operation and also contains a custom code section. See Appendix C for the list of Raduga built-in customizable steps.

## Defining Custom Actions

External steps defined for a list of entities appear in the entity's context menu as "Actions". Here is an example of the "ebs.CompileForm" step displayed as a "Compile Form" action in the ebs.Forms entity's context menu.




Users can select the "Compile Form" action to compile the server form.

To define a custom action, create an external step and choose the entities that are relevant to its action. The step will automatically appear in the context menu when the user selects an object of that entity.

# Entities

## Defining Entities

An Entity Type is an internal representation of a Raduga object. An Entity defines its object's presentation to the user. When the user chooses an entity, for example, ebs.Alerts, on the main Raduga form, Raduga shows a list of alerts defined in the current environment on the right panel.

Every Entity is connected to one Entity Type.

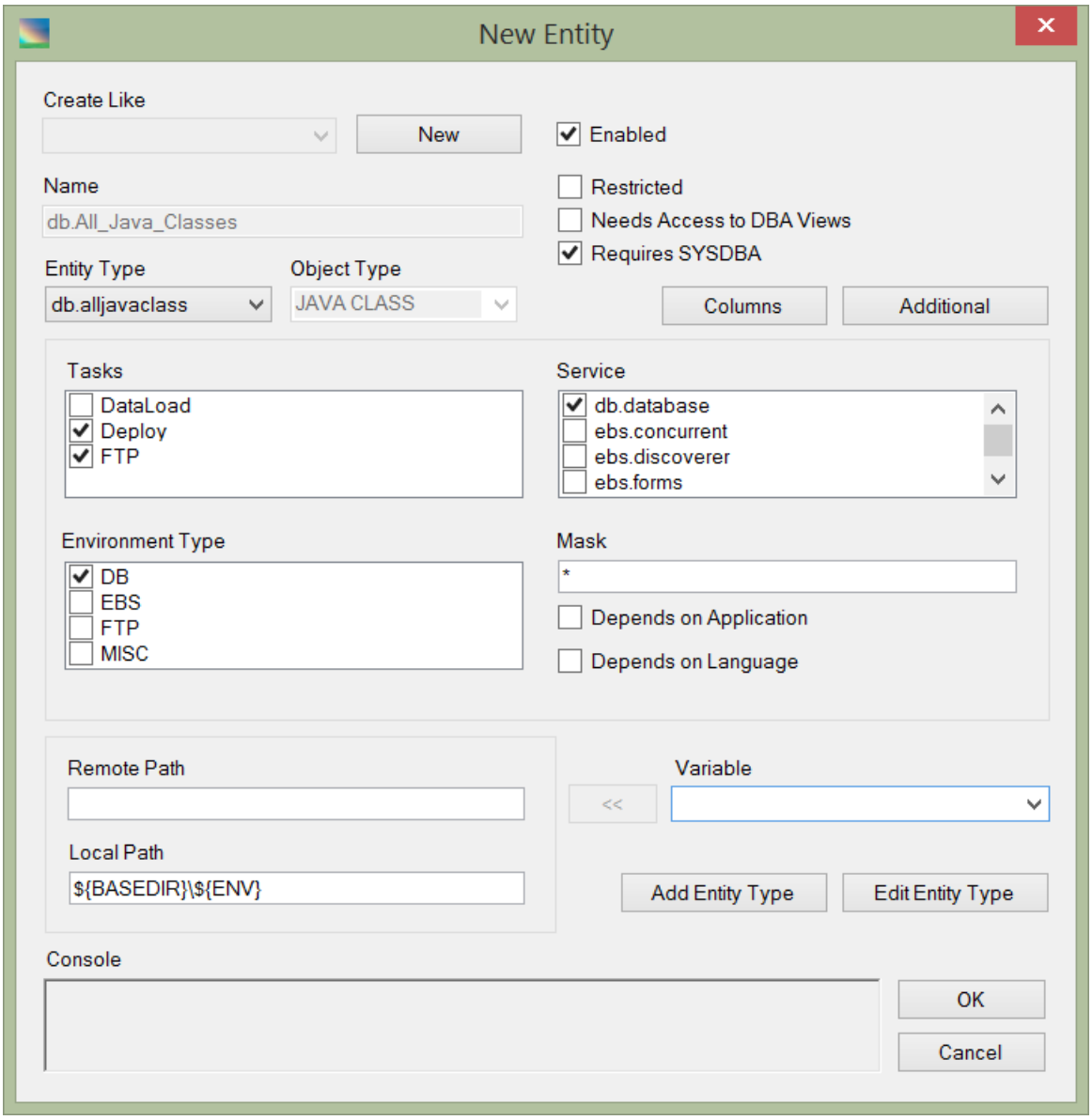
The configuration of an Entity is a set of rules that define how the Entity will be presented to the user and how its objects will be implemented on the server side. Here are some examples of entities:

db.Packages
ebs.Alerts
ebs.Profiles

The list of available entities appears when you choose "Entities" in the drop down box in the Raduga Global Configuration screen.

- To edit an entity, choose it in the "Entities" list box and press "Edit".
- To add new entity, press "Add".

The “Entity” definition window opens:




The following fields and controls are available:

**Create Like** Choose an entity from the drop down list to use as a template for the new entity.

**New** Press to create a new entity.

**Enabled** Select to enable the entity

**Name** Entity name.

**Entity Type** Entity type that is connected to the entity.

**Restricted** Restricted entities do not allow end users to change navigation paths and object masks.

| | |
|---|---|
| **Needs Access to DBA Views** | Select to require access to DBA views for the entity to deal with its objects.<br>The entity will not be available for environments that do not have DBA view access. |
| **Requires SYSDBA** | Select to require a sysdba connection for the entity to deal with its objects. |
| **Columns** | Select to define entity display columns (See “Defining Entity Columns”). |
| **Additional** | Select to define additional variables for the entity (See “Defining Additional Values”). |
| **Tasks** | A list of tasks for which the entity will be displayed. |
| **Environment Types** | Environment types (DB, EBS, FTP, MISC) for which the entity will be displayed. |
| **Service** | Services (db.database, ebs.concurrent …) that the entity objects belong to. |
| **Mask** | A default mask that restricts a list of objects displayed to the user. |
| **Depends on Application** | Select to make entity objects dependent on the application. The “Application” drop down list becomes available to the end user and the ${APP} variable is defined inside the entity definition. |
| **Depends on Language** | Select to make entity objects dependent on the language. The “Language” drop down list becomes available to the end user and the ${LANGUAGE} variable is defined inside the entity definition. |
| **Remote Path** | The entity’s default remote path. A remote path example (taken from the ebs.Framework entity):<br><br>$JAVA_TOP/${CUSTOM_APP}/oracle/apps/${APP_LC}<br><br>In this example:<br>${CUSTOM_APP} an additional variable that can be entered by the end user<br>${APP_LC} a Raduga built-in variable that contains the application name in lower case |
| **Local Path** | Entity’s default local path. The local path typical example is:<br><br>${BASEDIR}\${ENV}<br><br>In this example:<br>${BASEDIR} is a Raduga variable that contains the private working area path for the<br><br>${ENV} is a Raduga variable that contains the current environment name. |
| **Add Entity Type** | Click to open the empty entity type definition window. |

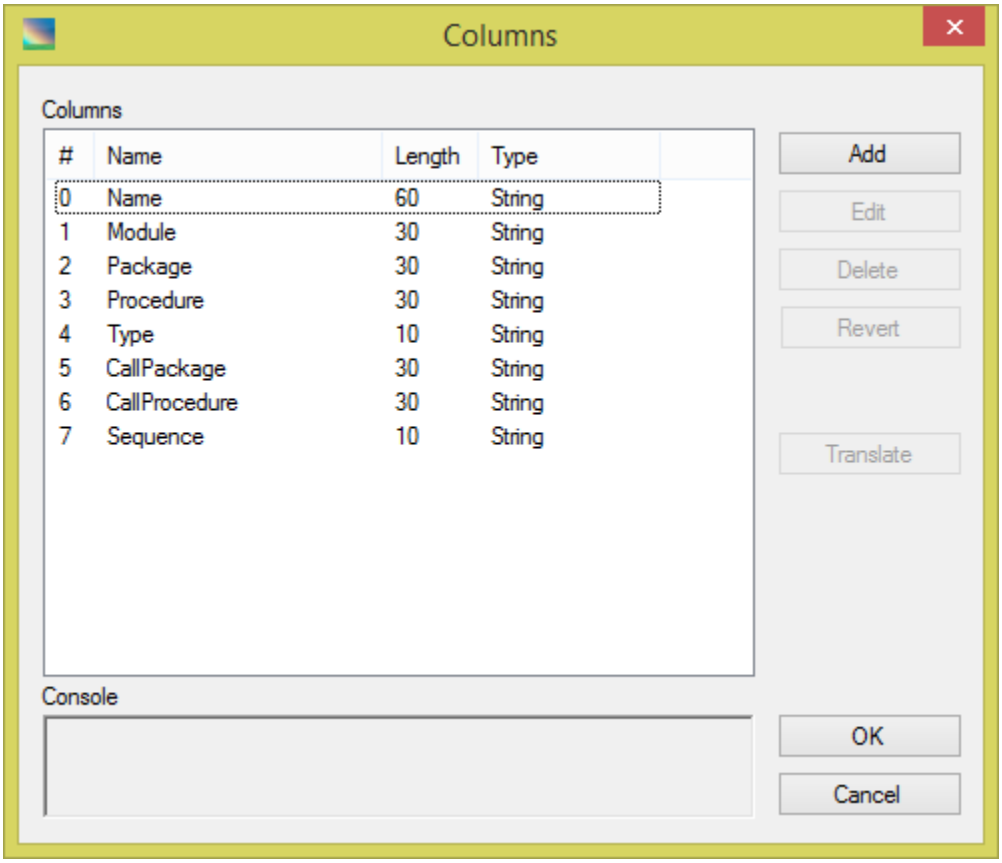
**Edit Entity Type** Click to open the entity type definition window.

## Defining Entity Columns

Entity default columns correspond to the main file attributes:

- Name
- Date
- Size

However, you can define custom columns for each entity. To do so, press “Columns” in the “Entity” form to display the “Columns” window:




You can select an existing column or add a new one. The form displays the following information for each column:

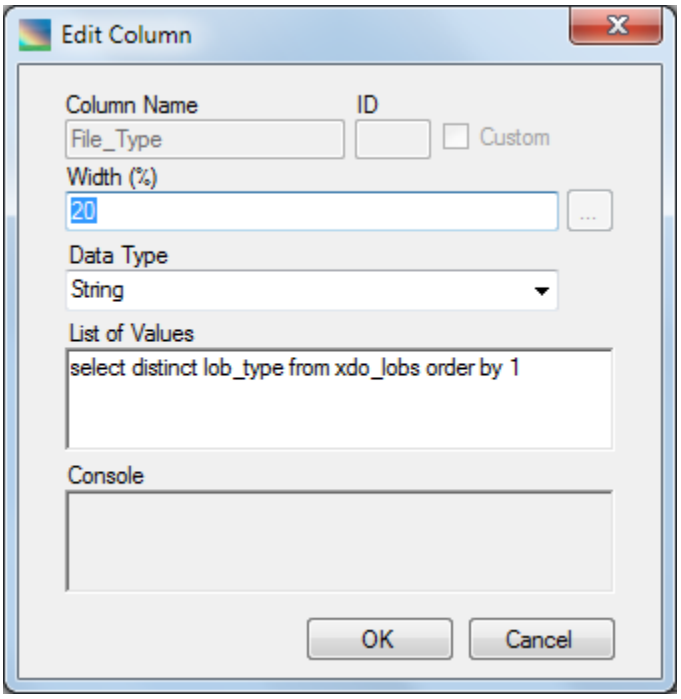
**#** Column’s sequential order

**Name** Column’s name (it is displayed to the end user if no translation for the column is defined)

**Length** Column’s length as a percentage of the right Raduga panel’s width

**Type** Column’s data type

Double click on one of the columns from the list to edit it. The “Edit Column” form appears:




The following fields are available:

**Column Name** Column’s name (it is displayed to the end user if no translation for the column is defined)

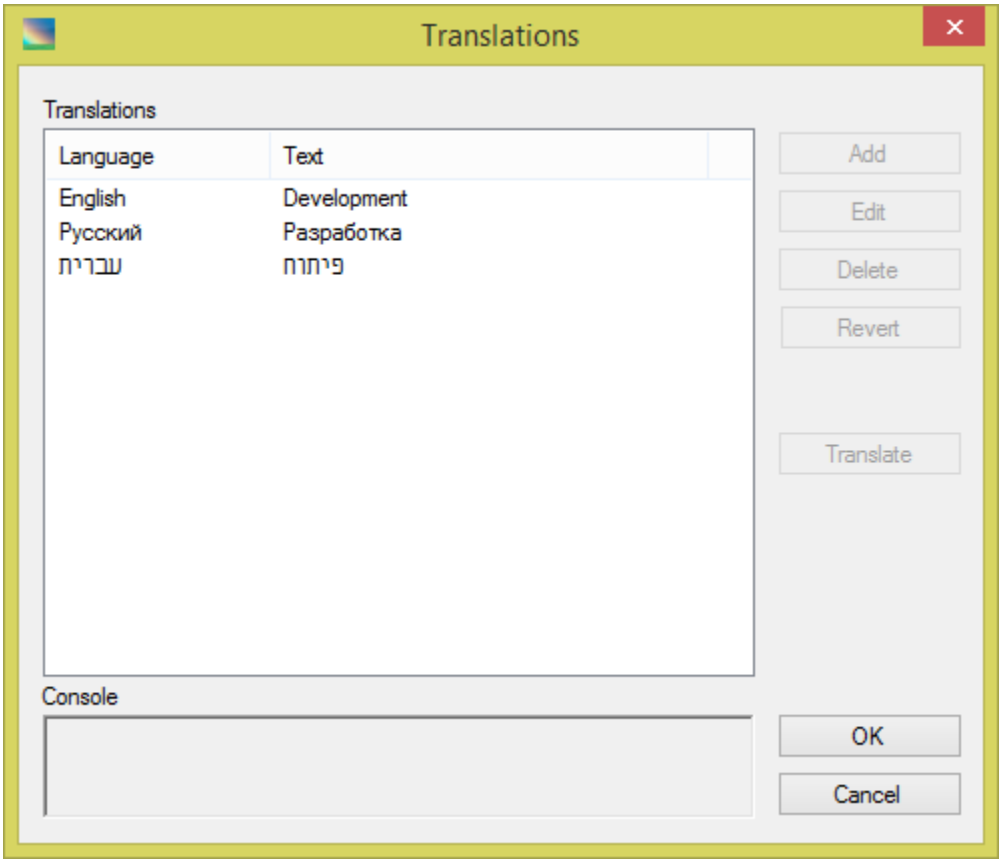
**Width** Column’s length as a percentage of the right Raduga panel’s width

**Data Type** Column’s data type

**List of Values** The column’s valid values (see section “Defining Valid Values”)

## Defining Column Translations

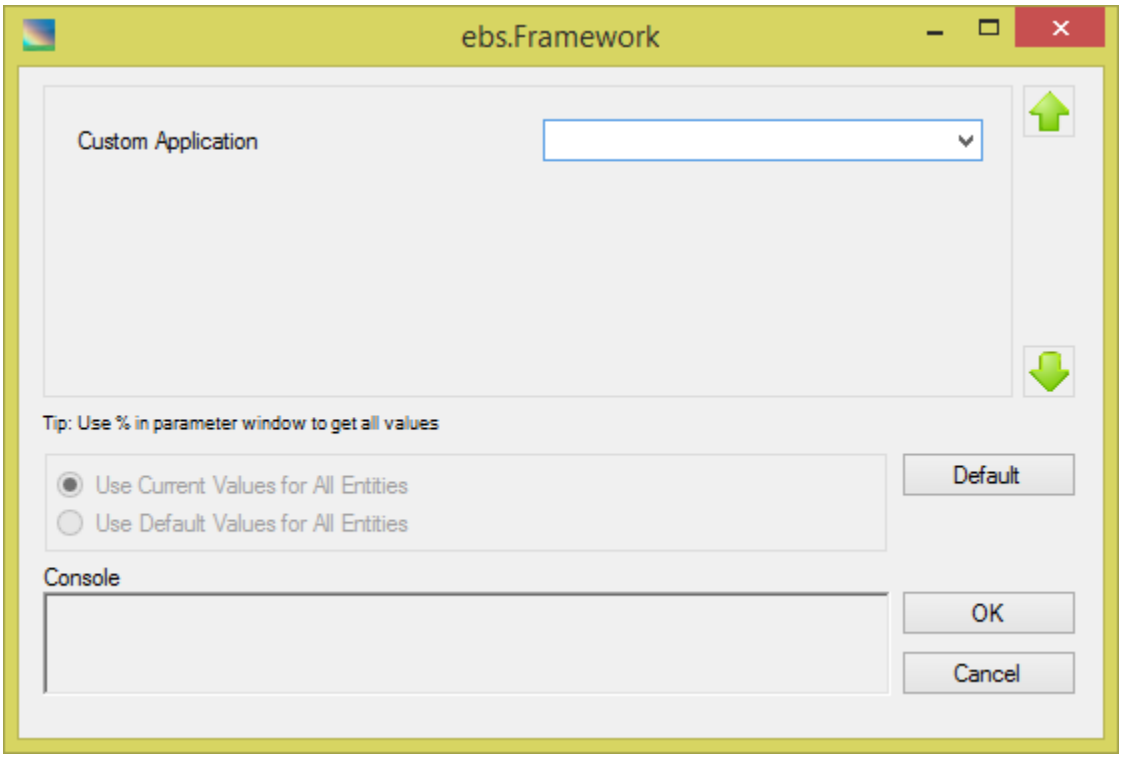
To display to the end user the translated caption for the column, you can define column's translation. Press "Translate" on the Columns to open the translations form:




Translate the column's caption for all available languages and press "OK"

## Defining Additional Values

You can use additional variables in the definition of entity type operations. For some entities it is necessary to collect additional information from the end user. If additional variables exist for the entity, the "+" sign to the right of local mask becomes available to the end user on the main Raduga window. Press the "+" sign to enter values for additional variables:




During the additional variable definition you can add a list of valid values which can be in a comma separated list of strings or a SQL statement that returns one column (See "Defining Valid Values"). The additional variable name can be used in the entity type operation definition scripts and in remote path definition. For example, the remote path definition for the ebs.Framework entity is:

$JAVA_TOP/${CUSTOM_APP}/oracle/apps/${APP_LC}

Here ${CUSTOM_APP} is a name of the additional variable that is entered by the end user. So the end user can alter the remote path according to the custom application selection.

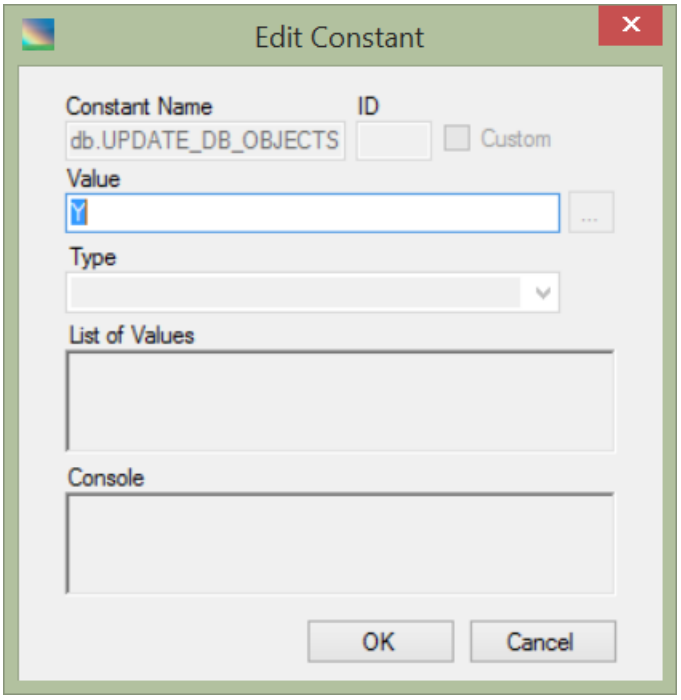
## Defining Constants

Constants are objects that hold constant values and influence application behavior. Constants can hold directory paths, date formats, yes/no values, etc.

- To edit a constant, open Global Configuration, choose "Constants" in the "Objects" drop down and press "Edit". Choose a constant in the "Constants" list box and press "Edit".

- To add new constant, press “Add”.

The “Constant” definition window appears:




In the “Edit Constant” form there are the following fields:

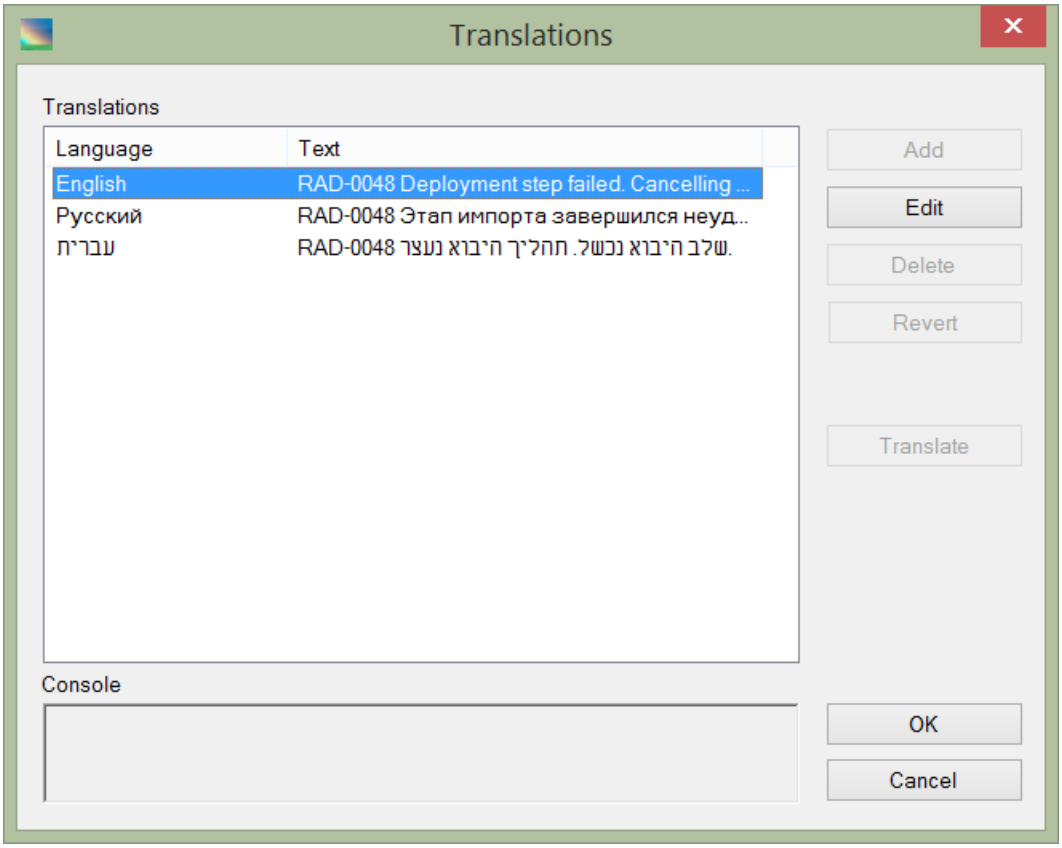
| | |
|---|---|
| **Constant Name** | The name of the constant |
| **Value** | The constant value |
| **ID**, **Type, List of Values** | These fields are always disabled for constants |

## Defining Messages

A Raduga administrator can customize messages that are displayed by the Raduga application.

To edit a message, open Global Configuration, choose “Messages” in the “Objects” drop down and select “Edit”. Choose a message in the “Messages” list box and select “Translate”.

The message “Translations” window appears:




You can edit the message and change its text in each listed language.

## Defining Services

Services are internal objects that connect entities and servers. In general, each server can host several services; for example, one server can be used to host the database and serve as a concurrent manager server. In this case it will host both db.database and ebs.concurrent services. On the other hand entities can belong to specific services. For example, the db.Sequences entity belongs to the db.database service and does not belong to the ebs.forms service. In this specific example, during the Raduga migration process, the db.Sequences entity objects will be migrated only to the servers that host db.database service.

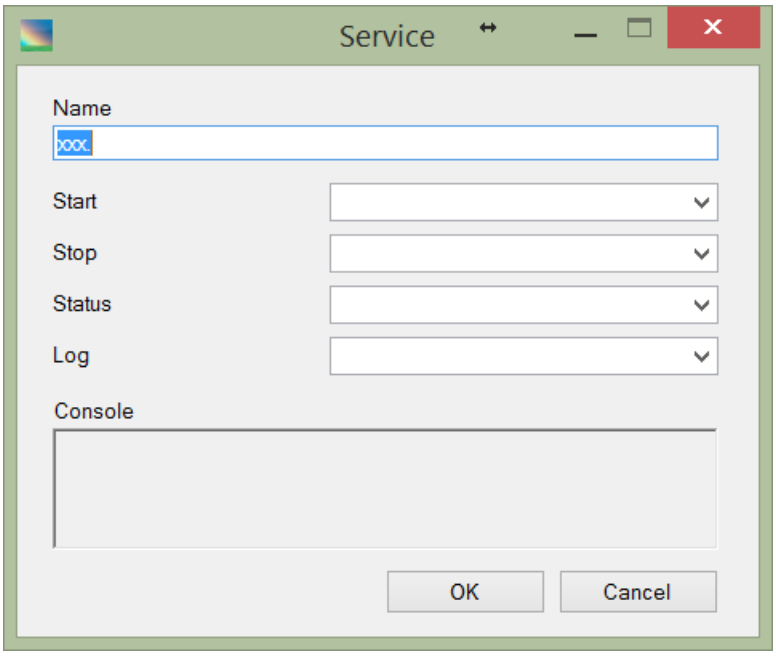
Raduga provides several built-in services:

db.database
db.listener
ebs.concurrent
ebs.mailer
ebs.discoverer
ebs.forms
ebs.framework
ebs.opmn
ebs.reports
ebs.web
ebs.discoverer
ebs.discoverer_infra
util.file

You can also add custom services. To add a service:

- Open Global Configuration
- Choose “Services” in the “Objects” drop down and press “Edit”.
- In the Services list form, press “Add”.

The “New Service” form appears:




The following fields are available:

| | |
|---|---|
| **Name** | The name of the service |
| **Start** | Select the procedure to start the service |
| **Stop** | Select the procedure to stop the service |
| **Status** | Select the procedure to check the service's status |
| **Log** | Select the procedure to view the service's log file |

Start, stop, status and log procedures are the existing Raduga steps you should create before creating the service. Raduga has procedures (steps) for starting, stopping, status checking and log file viewing for its built-in services. For example, to start the ebs.concurrent service, Raduga uses the ebs.cm_start step. These definitions are used for monitoring, starting and stopping Raduga environments.

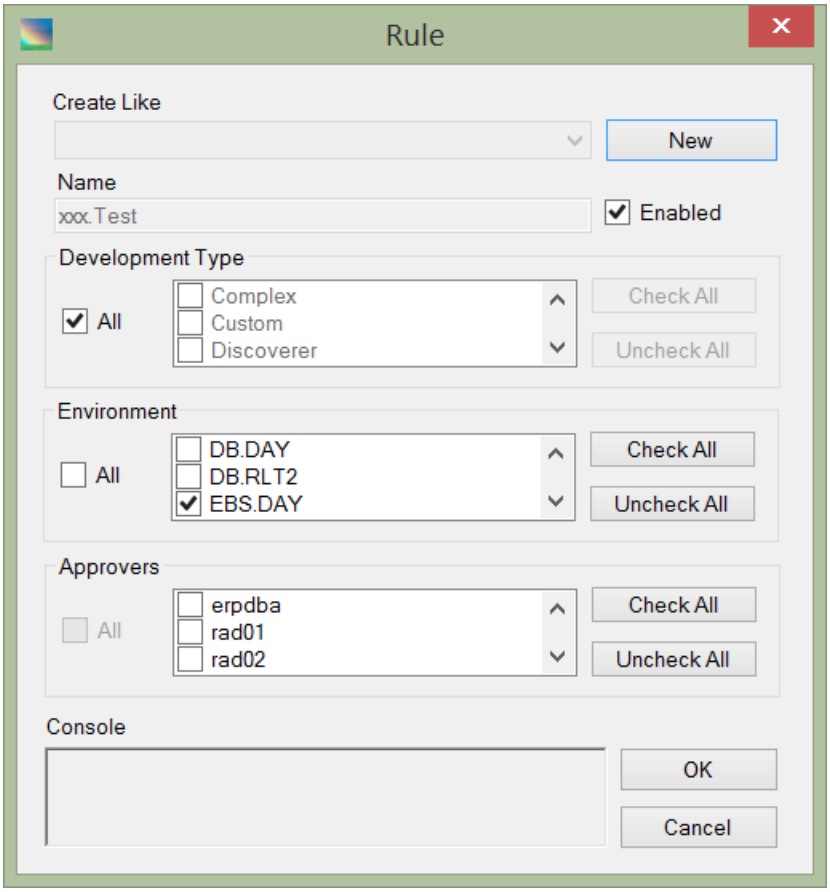
## Defining Project Approval Rules

Approval rules define the list of Raduga users who need to approve the development project for deployment to a specific environment. Raduga does not allow developers to deploy the project in an environment restricted by the Project Approval Rule if not all users defined by the rule approve it for deployment.

To add a rule:

- Open Global Configuration
- Choose "Project Approval Rules" in the "Objects" drop down and select "Edit".
- In the Project Approval Rules list form, select "Add".

The “Rule” form appears:




The following fields are available in the form:

**Name** The name of the approval rule.

**Enabled** Check/uncheck to enable/disable the rule.

**Development Type** The type of development project affected by the rule. Choose “All” to create a rule that affects all development projects. Choose specific development types to restrict the rule to those types.

**Environment** The environments affected by the rule. Choose “All” to require the project to be approved for deployment in all environments, or choose specific environments for which project deployment needs to be approved

**Approvers** A list of Raduga users comprising the approval chain. The project will not be approved for deployment unless all users on the approval chain approve it

# Reports

## Defining Reports

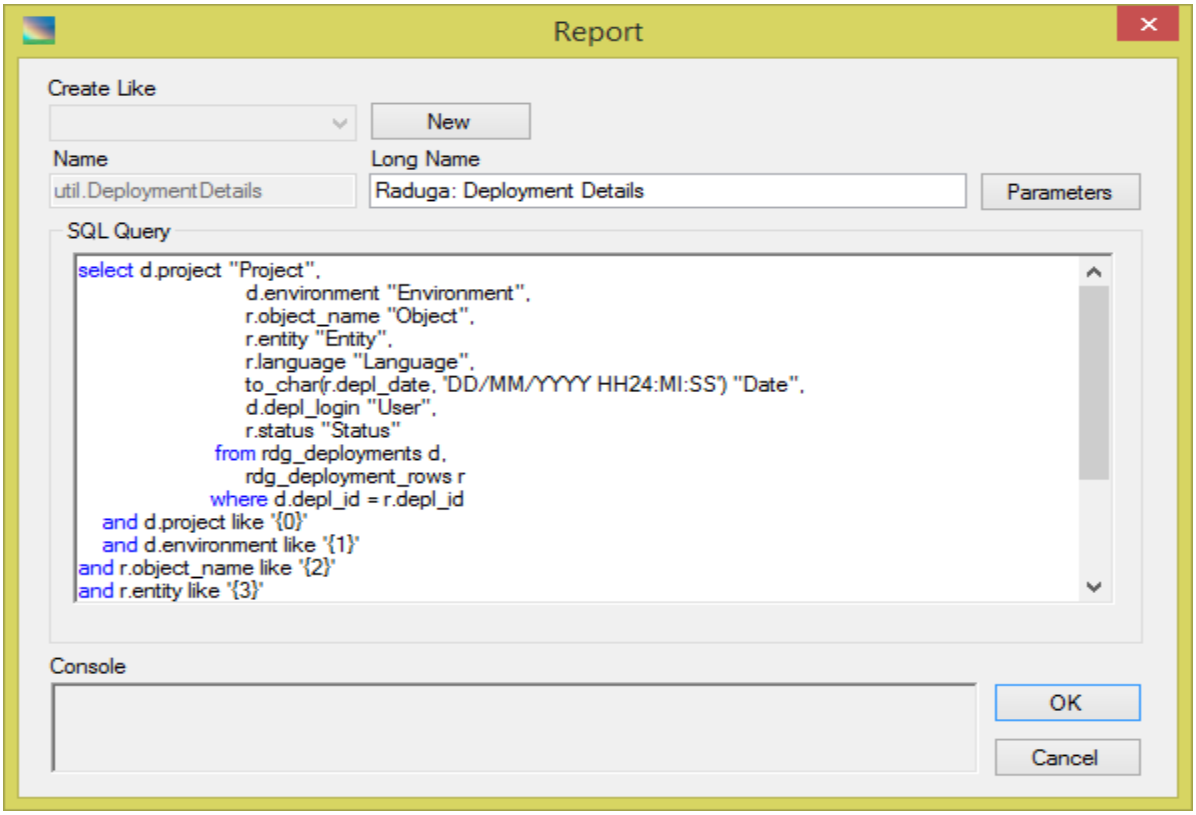
During Raduga configuration you can configure the Raduga Reporting Database. The database is not required for Raduga to function; however, you can use it to track activity. Once you have configured the Reporting Database, you can create reports that show Raduga objects usage history.

Raduga provides a set of pre-defined reports which you can use as-is or modify to suit your needs.

To edit existing report or create a new one:

1. Open Global Configuration.
2. Choose “Reports” in the “Objects” drop down and press “Edit”.
3. In the Reports list form, press “Add” to create a new report.
4. To edit an existing report, press “Edit”.

The “Report” window appears:




In the “Report” form there are the following fields and controls:

**Create Like** Choose a report from the drop down list to use as a template for a new report.

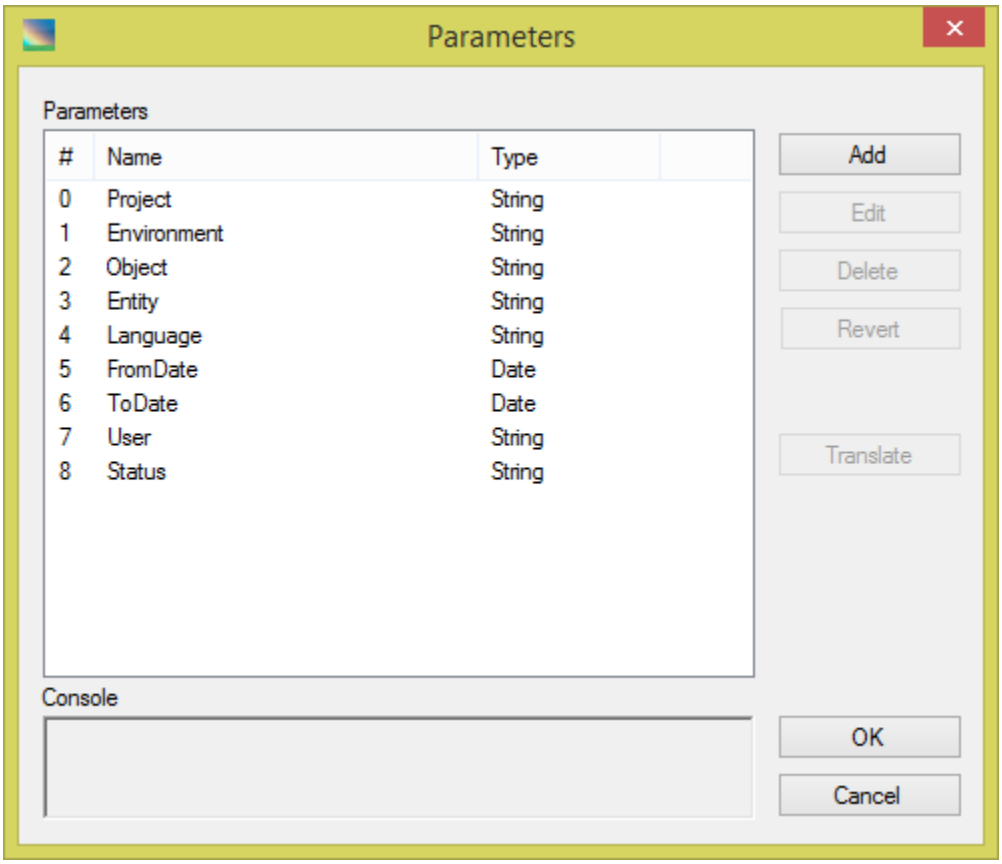
| | |
|---|---|
| **New** | Press to create a new report . |
| **Name** | The report name. |
| **Long Name** | The report name as it will be displayed to the end user. |
| **Parameters** | Press to define the parameters of the report. |
| **SQL Query** | The SQL that is used to produce the report's data. |

## Defining Report Parameters

In the SQL query used to produce the report's data you can add parameters in the following form: {N}, where N is an integer starting from 0. The parameters must be sequential integers without gaps. Here is an example of a SQL query with parameters:

```
select d.project "Project",
       d.environment "Environment",
       r.object_name "Object",
       r.entity "Entity",
       r.language "Language",
       to_char(r.depl_date, 'DD/MM/YYYY HH24:MI:SS') "Date",
       d.depl_login "User",
       r.status "Status"
  from rdg_deployments d,
       rdg_deployment_rows r
where d.depl_id = r.depl_id
   and d.project like '{0}'
   and d.environment like '{1}'
   and r.object_name like '{2}'
   and r.entity like '{3}'
   and (r.language like '{4}' or r.language is null)
   and ('{5}' is null or r.depl_date >= to_date(nvl('{5}',sysdate),'DD/MM/YYYY'))
   and ('{6}' is null or r.depl_date < to_date(nvl('{6}',sysdate),'DD/MM/YYYY')+1)
   and d.depl_login like '{7}'
   and r.status like '{8}'
 order by d.project, d.environment, r.depl_date
```

To define parameter properties, press “Parameters” to open the Parameters list form:




The “Parameters” form provides this information for each parameter:

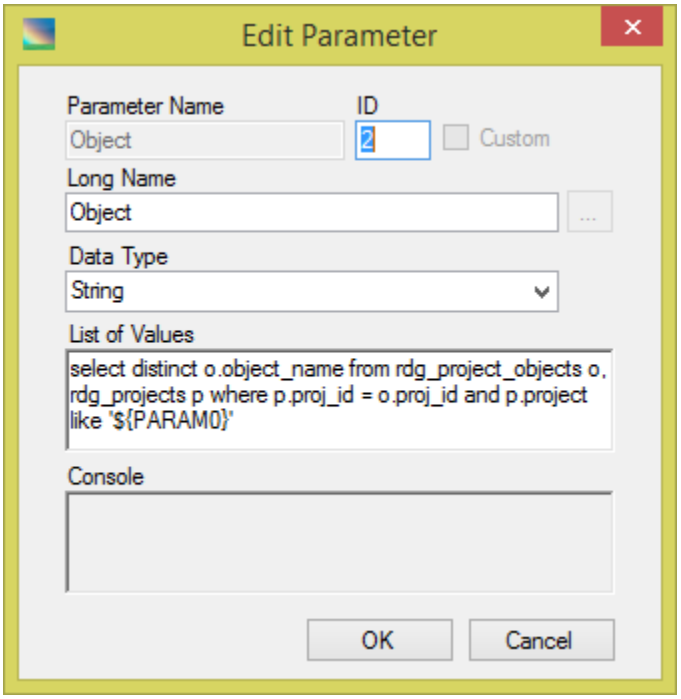
**Parameter Name** The name of the parameter

**ID** The parameter sequential Id

**Long Name** The parameter name displayed to the end user

**Data Type** The parameter data type (String, Date, Integer)

To edit an existing parameter, select the parameter and press “Edit”. To create a new parameter press “Add”. The “Edit Parameter form appears:




In the “Edit Parameter” form there are the following controls:

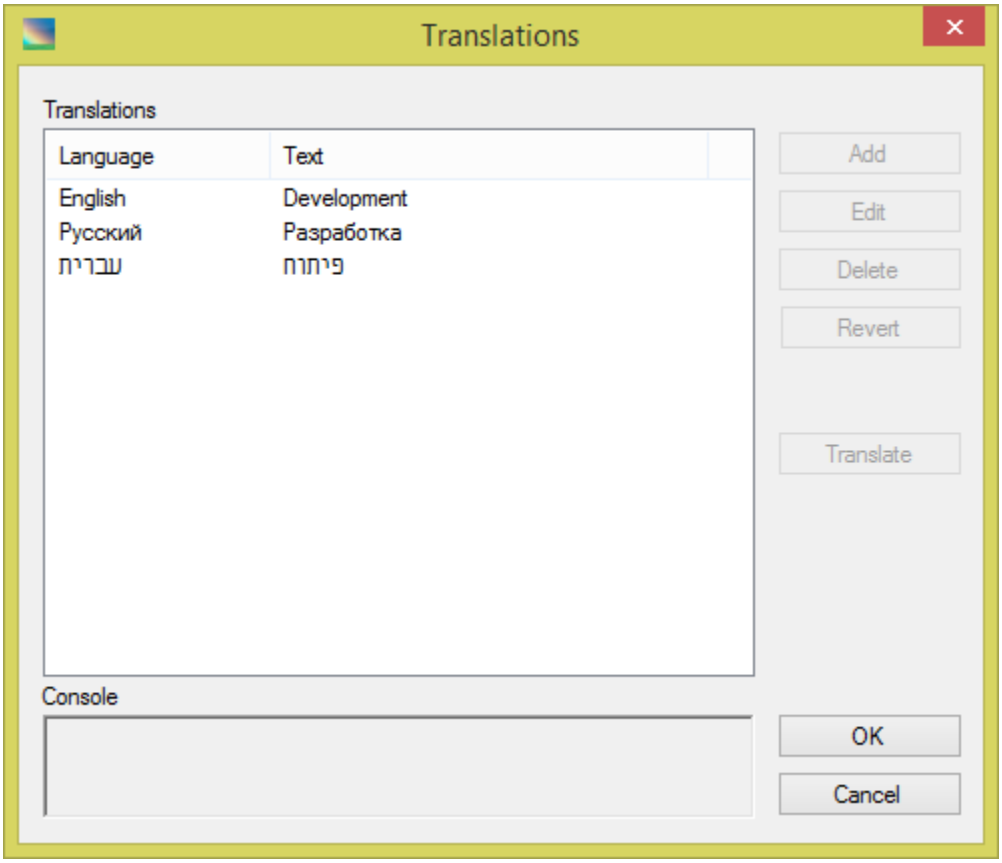
| | |
|---|---|
| **Parameter Name** | The name of the parameter |
| **ID** | The parameter sequential Id (it is used in the report SQL statement as {ID}) |
| **Long Name** | The parameter display name |
| **Data Type** | The parameter type |
| **List of Values** | The parameter’s valid values (see section “Defining Valid Values”) |

## Defining Parameter Translations

You can translate every report parameter into languages supported by Raduga. Currently there are three supported languages:

- English
- Hebrew
- Russian

To translate report parameters, select the parameter in the parameter list and press “Translate” to open the “Translations” form:




Provide a translation for the parameter display name in each supported language, then press “OK”.

## Development Steps

In general Raduga development consists of the following steps:

1. Describe the object you want to add to Raduga and create a functional design for the object's entity.
2. Add necessary constants, steps and services that the object's entity will use.
3. Create the Entity Type:
    a. Define "list", "get", "put" operations for the object.
    b. Optionally define "delete", "deploy" and "rename" operations for the object.
4. Create the Entity:
    a. Base the entity on the entity type you defined.
    b. Choose tasks, services, environments and a remote mask for the entity.
    c. Optionally define remote and local paths for the entity.
    d. Optionally define columns and additional variables for the entity.
5. Specify entity permissions. If necessary, assign the entity to specific users.

## Exporting and Importing Custom Objects

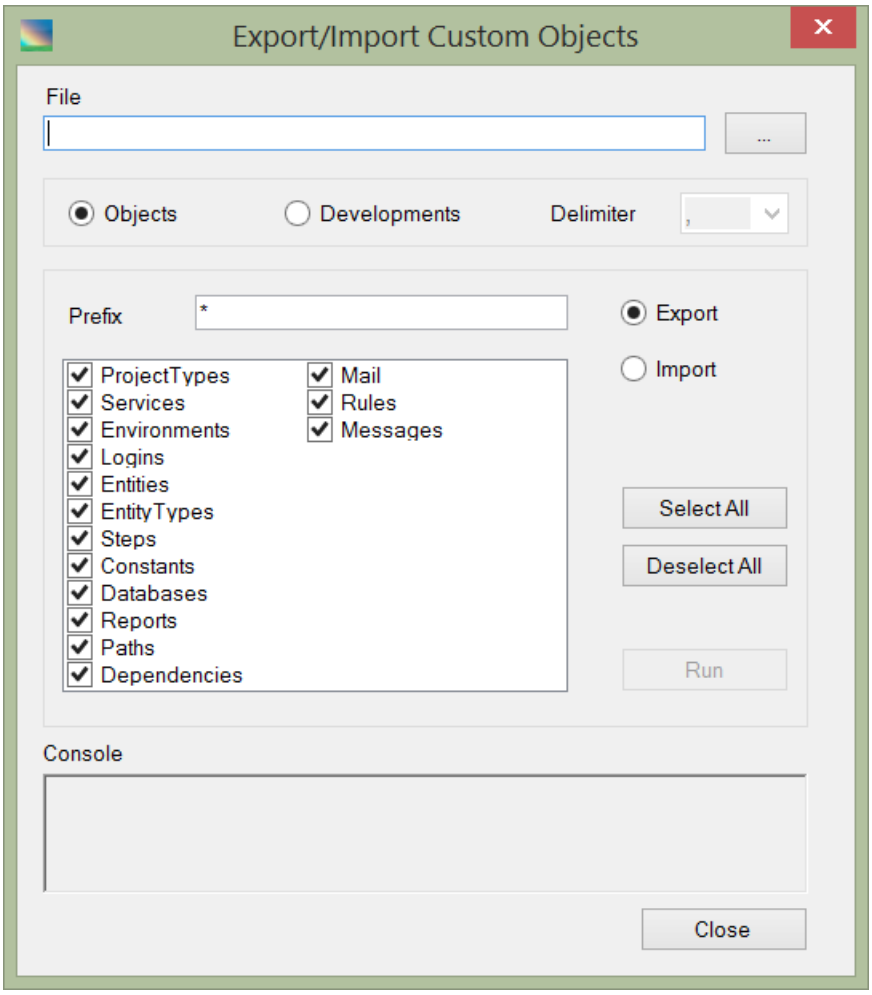
Raduga has a special interface for exporting and importing custom objects. Note that during the export process only customized objects (from the Raduga_Custom.xml configuration file) are actually exported.

Here is a list of Raduga custom objects that can be exported or imported:

Project Types
Services
Environments
Logins
Entities
Entity Types
Steps
Constants
Databases
Reports
Paths
Dependencies
Mail
Rules
Messages

There is an additional interface that allows importing Development Projects.

To export or import Raduga objects, open Global Configuration and select “Export Objects”. The “Export/Import Custom Objects” window appears:




In this window you can choose either Raduga Objects or Raduga Development Projects.

If you choose Raduga Objects:

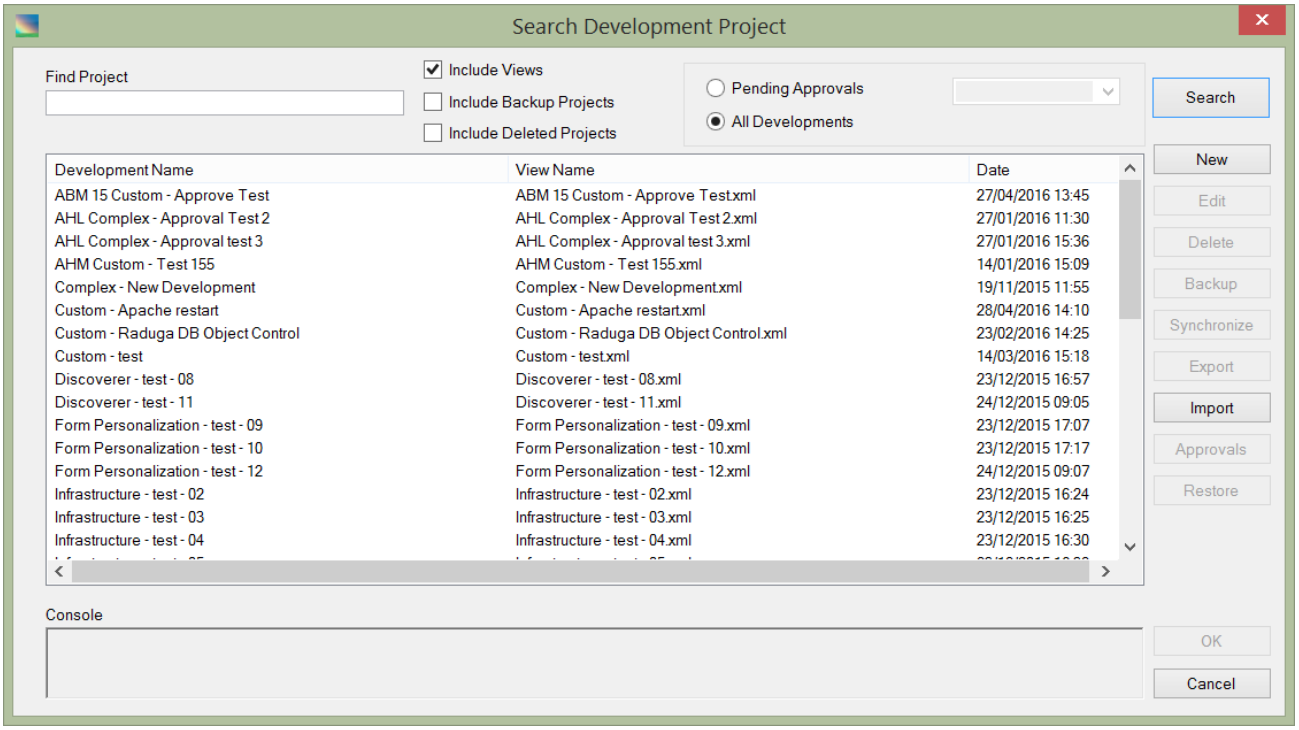
1. Select an export/import XML file.
2. Select an action (export or import).
3. If you choose export, fill the prefix text box (only objects with this prefix will be exported). Use “*” to export objects with any prefix.
4. Check the check box beside each object you want to import or export.
5. Click “Run”.

If you choose Developments, you can only import historical development projects. The import file should be a CSV file with fields separated by the character set in the Delimiter text box. The file structure should be as follows:

| Field | Type | Valid Values |
|---|---|---|
| Source | String | New, Migration |
| ExternalSystem | String | |
| ExternalID | String | |
| PrevID | String | |
| Name | String | |
| Description | String | |
| Module | String | Valid application name |
| ProjectType | String | Complex, Custom, Discoverer, Form Personalization, Interface, OAF, OAF Personalization, Other, Report, Workflow |
| MainOwner | String | Valid Raduga user |

After you import an historical development project, it will be created in Raduga. However, the development project will be empty. You need to manually add specific objects to the project.

Entire development projects can be exported/imported using the "Search Development Project" window (see "Opening an Existing Development Project" in the Raduga User Guide):




To export a development project, select it in the projects list and click "Export". The export ZIP file will be created. This file contains a project metadata file as well as all custom objects included in the project. This file can be used for importing the project into another Raduga instance.

In order to import the project, click "Import" and choose the project export ZIP file to import.

# Appendix

## Appendix A – List of Raduga built-in variables

| | |
|---|---|
| **APP** | Current application name |
| **APP_LC** | Current application name in lower case |
| **APPLOGIN** | Current application login name (can be different from Raduga login name) |
| **APPSERVER** | Current application server name |
| **APPSPASSWORD** | APPS password |
| **APPSUSER** | APPS user name |
| **BASEDIR** | Private base directory. For example: \\server\Raduga\<br>In this case user objects are stored under the following directory:<br>\\server\Raduga\users\<user>\developments\<environment>\<entity>\... |
| **CMSERVER** | Concurrent Manager Server (if there are 49ultiple CM servers, the first one) |
| **CONTEXT_FILE** | Application Context File |
| **CONTEXT_NAME** | Application Context Name |
| **DBCREDENTIALS** | A string that can be used to connect to the database. Example:<br>sqlplus ${DBCREDENTIALS} |
| **DBPORT** | Current database port |
| **DBSERVER** | Database server (if there are multiple database servers, the first one) |
| **DISCOSERVER** | Discoverer server |
| **ENTITY** | Current Entity |
| **ENV** | Current Environment |
| **ENV_FILE** | Current Environment configuration file |
| **EULPASSWORD** | Discoverer EUL password |
| **EULUSER** | Discoverer EUL user |
| **EXTENSION** | Current object extension |
| **FAILURE** | Current operation failure message |

| | |
|---|---|
| **FORMSSERVER** | Forms server (if there are multiple forms servers, the first one) |
| **LANGUAGE** | Current application language |
| **LICENSE** | Current Raduga user's license number |
| **LOCALDIR** | Current Local directory |
| **LOGIN** | Current Raduga login name |
| **LOGIN_SC** | Current Raduga login name in lower case |
| **MASK** | Current mask for server objects |
| **NEW_OBJECT** | New object name (during rename) |
| **NEW_OBJECT_HASH** | New object hash value |
| **NEW_OBJECT_NOEXT** | New object name without extension |
| **OBJECT** | Current object name |
| **OBJECT_<N>** | Text from the N'th column of the right Raduga panel (starting from 1) |
| **OBJECT_HASH** | Current object hash value |
| **OBJECT_NOEXT** | Current object name without extension |
| **PROJECT** | Current Project name |
| **REMOTEDIR** | Current Remote Directory |
| **REPORTSSERVER** | Reports Server (if there are multiple reports servers, the first one) |
| **REVISION** | Current object's revision number |
| **SERVER** | Current Server |
| **STAGE** | Raduga Stage Directory |
| **SUCCESS** | Current operation success message |
| **SYSTEMPASSWORD** | SYSTEM password |
| **TASK** | Current Task (FTP, Deploy or DataLoad) |
| **TOP** | Current Application Top directory |
| **USER** | Current OS user |
| **WEBSERVER** | Web Server (if there are multiple web servers, the first one) |

## Appendix B – List of Raduga built-in steps

| | |
|---|---|
| **db.get_udump_dir** | Set UDUMP variable to the value of user_dump_dest database parameter |
| **db.format_sql** | Break long SQL lines (more than 2499 characters) in current object definition |
| **db.get_user_grants** | Create SQL statement for granting privileges for current object (current user has no SYSDBA role) |
| **db.get_dba_grants** | Create SQL statement for granting privileges for current object (logged in as SYSDBA) |
| **ebs.check_fnd_status** | Parse FNDLOAD logfile (defined in the $logfile variable) and set MESSAGE and STATUS variables if failure occurs |
| **ebs.get_forms_url** | Set FORMS_URL variable according to the value taken from ICX_FORMS_LAUNCHER profile<br>Forms URL contains also “lang”, “NLS_LANG”, “NLS_DATE_LANGUAGE”, “FORMS_USER_DATE_FORMAT” and “FORMS_USER_DATETIME_FORMAT” values set according to the current NLS and Raduga definitions |
| **ebs.get_fnd_message** | Get FNDLOAD success message in the current application language |
| **ebs.get_fnd_errors** | Get FNDLOAD error messages in the current application language |
| **ebs.get_iso_lang** | Set ISO_LANG variable according to the application language |
| **ebs.get_nls_lang** | Set NLS_LANG and UPLOAD_MODE variables according to the current application anguage<br>NLS_LANG is constructed using NLS values taken from the database and has a format LANGUAGE_TERRITORY.CHARACTERSET<br>UPLOAD_MODE is used by FNDLOAD utility. It can be:<br>○ UPLOAD_MODE=REPLACE if current application language is US (base language)<br>○ UPLOAD_MODE=NLS if current application language is other than US (not base language) |
| **ebs.sync_wf_tables** | Synchronize workflow data in WF tables |
| **ebs.update_who_fields** | Update LAST_UPDATE_DATE and OWNER fields during FNDLOAD operation |
| **util.Failure** | Set failure status and build error messages |

| | |
|---|---|
| **util.find_files** | List server files to standard output (including subdirectories) |
| **util.list_files** | List server files to standard output (only current directory) |
| **util.rcs_break_rcs** | Break RCS the lock of the current object |
| **util.rcs_ci_rcs** | Check in the current object to the Raduga RCS repository |
| **util.rcs_co_rcs** | Check out the current object from the Raduga RCS repository |
| **util.rcs_def_rcs** | Define the Raduga RCS directory and owner for the current object |
| **util.rcs_log_rcs** | Print RCS history for the current object to standard output |
| **util.Success** | Set success status and build success messages |

## Appendix C – List of Raduga built-in customizable steps

| | |
|---|---|
| **ebs.apache_start** | Start Apache server (for E-Business Suite) |
| **ebs.apache_stop** | Stop Apache server (for E-Business Suite) |
| **ebs.apache_status** | Apache server status (for E-Business Suite) |
| **ebs.apache_log** | View Apache server log file (for E-Business Suite) |
| **ebs.app_start** | Start all application services (for E-Business Suite) |
| **ebs.app_stop** | Stop all application services (for E-Business Suite) |
| **ebs.cm_start** | Start concurrent manager (for E-Business Suite) |
| **ebs.cm_stop** | Stop concurrent manager (for E-Business Suite) |
| **ebs.cm_status** | Concurrent manager status (for E-Business Suite) |
| **ebs.cm_log** | View internal concurrent manager log file (for E-Business Suite) |
| **ebs.oacore_start** | Start framework agent (for E-Business Suite) |
| **ebs.oacore_stop** | Stop framework agent (for E-Business Suite) |
| **ebs.oacore_status** | Framework agent status (for E-Business Suite) |
| **ebs.oacore_log** | View framework agent log file (for E-Business Suite) |
| **ebs.forms_start** | Start forms server (for E-Business Suite) |
| **ebs.forms_stop** | Stop forms server (for E-Business Suite) |
| **ebs.forms_status** | Forms server status (for E-Business Suite) |
| **ebs.forms_log** | View forms server log file (for E-Business Suite) |
| **ebs.mailer_start** | Start Notification Mailer (for E-Business Suite) |
| **ebs.mailer_stop** | Stop Notification Mailer (for E-Business Suite) |

| | |
|---|---|
| **ebs.reports_start** | Start reports server (for E-Business Suite 11i) |
| **ebs.reports_stop** | Stop reports server (for E-Business Suite 11i) |
| **ebs.reports_status** | Reports server status (for E-Business Suite 11i) |
| **ebs.reports_log** | View reports server log file (for E-Business Suite 11i) |
| **ebs.compile_apps** | Compile APPS schema (for E-Business Suite) |
| **ebs.compile_jsp** | Compile JSP (for E-Business Suite) |
| **util.define_env** | Source environment file. If it does not exist – source .bash_profile or .profile. |
| **util.rcs_break** | Break the RCS lock of the the current object |
| **util.rcs_ci** | Check in the current object to the Raduga RCS repository |
| **util.rcs_co** | Check out the current object from the Raduga RCS repository |
| **util.rcs_def** | Define the Raduga RCS directory and owner for the current object |
| **util.rcs_log** | Print RCS history for the current object to standard output |
| **util.Exit** | Exit Raduga script |

## Appendix D – List of Raduga built-in constants

| Constant | Default Value | Description |
|---|---|---|
| db.UPDATE_DB_OBJECTS | N | Allow to drop or rename database objects |
| db.DROP_OBJECT_BEFORE_CREATE | N | If the value of this constant is 'Y', then for database tables, users, tablespaces and sequences the database object will be dropped as a part of new object creation. |
| ebs.UPDATE_EBS_OBJECTS | N | Allow to drop or rename E-Business Suite objects |

| | | |
|---|---|---|
| ebs.UPDATE_WHO_FIELDS | Y | Allow to set owner and last update date during the migration of E-Business Suite objects |
| ebs.FORMS_USER_DATE_FORMAT | DD%2FMM%2FRRRR | FORMS_USER_DATE_FORMAT variable |
| ebs.FORMS_USER_DATETIME_FORMAT | DD%2FMM%2FRRRR+HH24%3AMI%3ASS | FORMS_USER_DATETIME_FORMAT variable |
| ebs.DISCO_ADMIN_RESPONSIBILITY | System Administrator | E-Business Suite responsibility that is used by Discoverer for administrative tasks |
| ebs.JAVA_COMPILER | javac | Default java compiler for compiling java source code |
| ebs.JAVA_DECOMPILER | jad -p | Default java decompiler for decompiling java classes |
| util.UPDATE_FS_OBJECTS | N | Allow to drop or rename file system objects |
| util.CUSTOM_PREFIX | xxx | Default prefix for custom objects |
| util.VERSION_CONTROL_SYSTEM | rcs | Version control system used by Raduga |
| util.BROWSER | iexplore.exe | Default internet browser |
| util.ENCRYPT_OBJ_PROPS | N | If the value of this constant is 'Y' then the content of the property file for Raduga objects will be encrypted |
| util.PASSWORD_DAYS | 0 | User password expiration period in days. If 0 - the password will never expire |
| util.PROJ_LOCKED_BY_APPROVER | N | If the value of this constant is 'Y', the development project will be locked for changes if it is approved for deployment. |
| util.REQUIRE_REPDB_FOR_ENV | | A list of environments that require Reporting database to be available. If the reporting database is not available, deployment operations are not allowed for these environments. |
| cloud.HCM_RESOURCES_PATH | | A path that must be concatenated to the Cloud URL to obtain human resources. For example: hcmRestApi/resources/11.13.18.05 |

| | | |
|---|---|---|
| cloud.HCM_HCIM_PATH | | A path that must be concatenated to the Cloud URL to obtain human resources users. For example: hcmRestApi/scim |
| PRIVATE_CONFIG_DIR | | User preferences base directory |
| PRIVATE_WORKING_DIR | | User objects base directory |
| PROJECTS_DIR | | Development projects base directory |
| RDG_ADMINISTRATOR_GRP | | Raduga administrators windows group name |
| RDG_DEVELOPER_GRP | | Raduga developers windows group name |
| RDG_IMPLEMENTER_GRP | | Raduga implementers windows group name |
| RDG_USER_GRP | | Raduga users windows group name |

## For Further Information

For any questions regarding this product, contact us at support@LazyDeploy.com, tel. +79185402272, or visit Raduga's web site: http://www.LazyDeploy.com