

Using the Design Explorer

Christian Vecchiola and Xingchen Chu

Abstract

This tutorial describes the Aneka Design Explorer and explains how to quickly prototype parameter sweeping applications that runs on the Aneka Cloud. It illustrates the features of the user interface environment shipped with the Design Explorer and provides a step by step guide on how to compose applications and monitor their execution on the Aneka Cloud. After having read this tutorial the users will be able to develop their own parameter sweeping applications with the Design Explorer.

Document Status

Creation Date:	08/28/09
Version:	0.1
Classification:	User
Authors:	Christian Vecchiola, Xingchen Chu
Last Revision Date:	08/28/09
Status:	Draft

1. Prerequisites

In order to fully understand this tutorial the user should be familiar with the general concepts of Grid/Cloud Computing and the XML language.

The practical part of the tutorial requires a working installation of Aneka. The common Aneka distribution contains the Design Explorer.

2. Introduction

The Design Explorer is an integrated environment that allows user to quickly prototype distributed applications based on the *Parameter Sweeping* model. Users can easily identify the logic and the data of the distributed application by using a sequence of steps that guides them in composing the distributed application. In this tutorial we will: first and then illustrate the features of the Design Explorer

- *characterize the nature of the Parameter Sweeping applications*

- illustrate the features of the Design Explorer
- provide a step by step guide on how to create an application with the Design Explorer

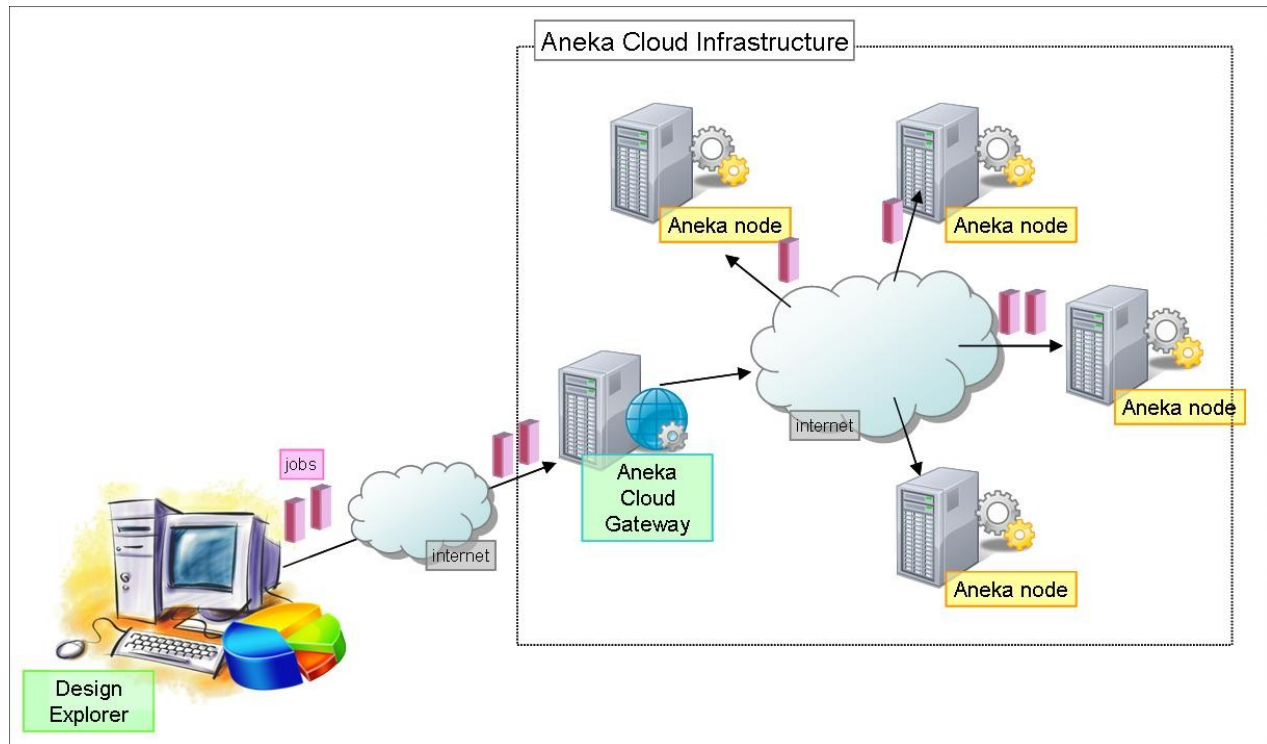


Figure 1. Application Scenario.

Figure 1 describes the common scenario in which the Design Explorer is used. A user interact with the user interface provided with the Design Explorer, composes the parameter sweeping application and then submits the collection of jobs that represent the application to the Aneka Cloud. By using the same interface the user can control the execution of the application and control the status of the jobs.

The Design Explorer can also work in stand-alone mode. In this case it is only possible to *compose* the parameter sweeping application. For executing it, it is necessary a live connection to the Aneka Cloud.

3. Parameter Sweeping Applications

3.1 Definition and Characteristics

A parameter sweeping application is a kind of distributed application that is defined by a *template task* characterized by a set of *configurable parameters*. The template task identifies the set of operations that define the computation. The configurable parameters represent the way in which the template task is specialized. Each of these parameters could have a different domain and users want to explore the behavior of

the template task for all the possible values of the parameters. The exploration of the parameter values generates a variable number of tasks representing the jobs of the parameter sweeping application. Every single job represents an executable entity with a specific parameter setting.

Parameter sweeping applications are quite common in different areas such as scientific computing and finance. They represent the most intuitive way to provide distributed support for legacy applications designed to run on a single machine.

For example, in the field of scientific computing the SETI project can be¹ considered as a kind of parameter sweeping application. SETI (Search for Extra Terrestrial Intelligence) @home is a project aimed at detecting intelligent life outside Earth by analyzing the radio frequency signals coming from the space. The range of signals to explore, the observation time, and the portion of the sky covered make up a huge amount of data to analyze. This data is divided into chunks that can be analyzed in parallel by the same application. In this case the template task is represented by the application and the configurable parameters identify the specific chunk of data to analyze.

In general there exist a large number of legacy applications that are controlled by a set of parameters. All these applications, can take advantage of the parameter sweeping model in order to distribute the execution and explore the entire parameters domain in a more effective way.

3.2 Example

Figure 2 describes the process of generating the jobs from a template task that is characterized with parameters. A common template task can be composed by the following elements:

- One or more executable applications that define the sequence of operations that are performed by the template task. Parameters representing the variable elements in the template tasks that specialize its behavior. The parameters can characterize different elements such as: command switches, input and output file names, and also file content.
- Input files. They can be data files, configuration files, or executable applications.
- Output files. They generally are the outcome of the computation of the template task as a whole.
- Parameters representing the variable elements in the template tasks that specialize its behavior. The parameters can characterize different elements such as: command switches, input and output file names, and also file content.
- Input files. They can be data files, configuration files, or executable applications.
- Output files. They generally are the outcome of the computation of the

¹ More information can be found at: <http://setiathome.berkeley.edu/>

template task as a whole.

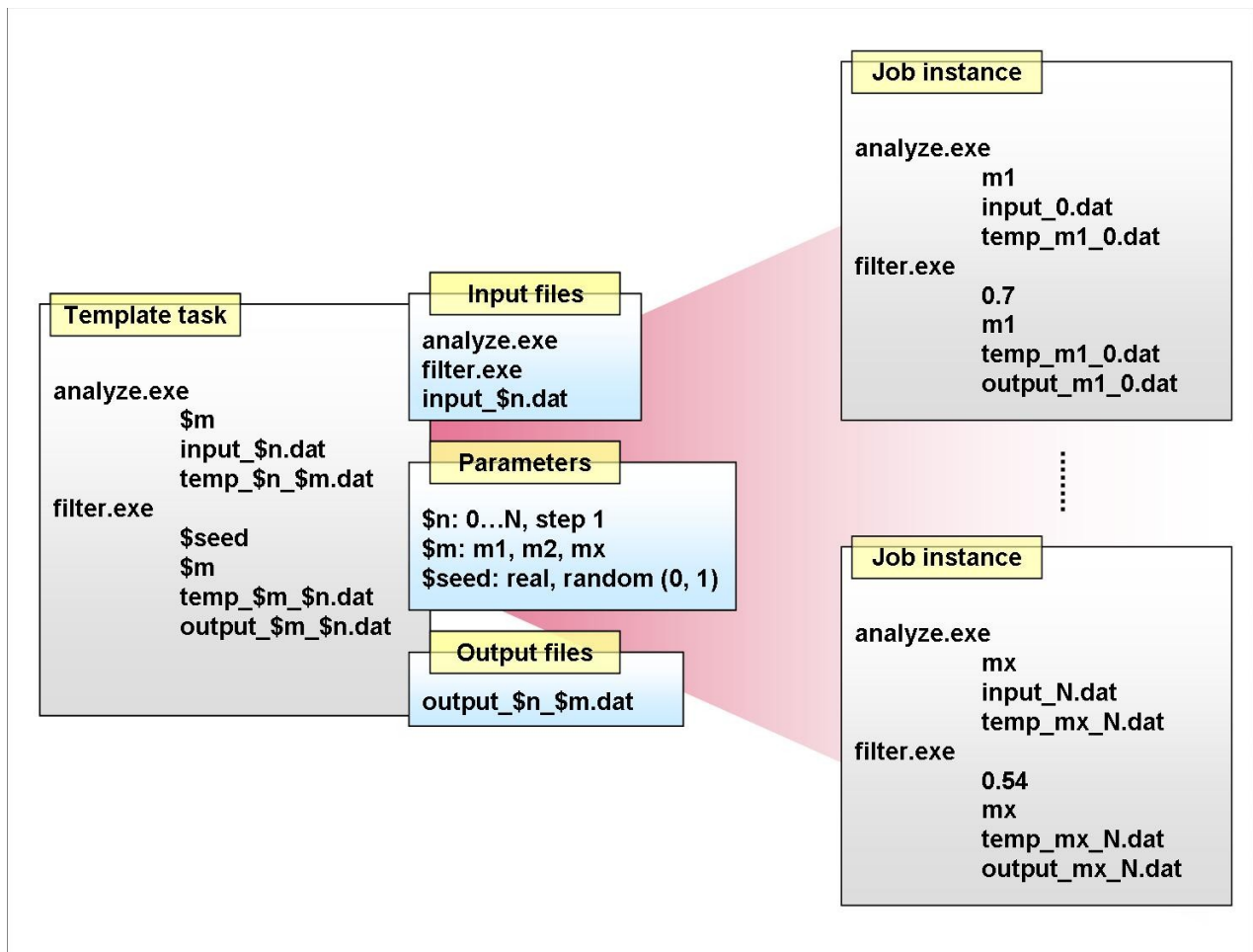


Figure 2. Job Generation from a Template Task.

The number and the specific domains of the parameters determine the number of jobs that compose the parameter sweeping application. In the example shown in Figure 2, a simple data analysis application is considered. The template task runs two console application in sequence and it is controlled by three different parameters. It takes three input files and produces one output file. The output of the task is the result of the chained execution of *analyze.exe* and *filter.exe* as described in the template task. As shown in the figure, *analyze.exe* takes as input a command switch identified by the $\$m$ parameter, the file *input_\${n}.dat* and produces the output *temp_\${m}_\${n}.dat*. The filter application takes as input the command switch $\$m$, the *temp_\${m}_\${n}.dat* file previously generated and a random seed number identified by the $\$seed$ parameter. It produces the output files *output_\${m}_\${n}.dat* representing the outcome of the task.

The scenario that this parameter sweeping application explores is identified by all the possible combinations of the the two parameters $\$n$ and $\$m^2$. The former identifies the specific chunk of data processed while the latter represents the specific processing

2 The random parameter $\$seed$ is not considered in the scenario since it does not define a dimension of the problem but simply represents an random runtime value used to initialize the filter.exe application.

mode used. The set of all the combinations can be expressed as

$$C: [0, \dots, N] \times [m_1, m_2, \dots, m_x]$$

and generates a number of task that is equal to $3 \times (N+1)$. For each of the points that belong to the scenario C a specific job is generated where the occurrences of all the parameters defining the scenario are substituted by the corresponding parameter values. In the example considered, there also exists a random parameter $\$seed$ that does not belong to the scenario but it is simply generated at run time. These jobs are then executed and the results are collected.

3.3 Parameter Sweeping Support within Aneka

Different distributed infrastructure provide different run time support for parameter sweeping applications. For this reason, while parameter sweeping applications are a general model there exist no standard language or format to represent the template task and the parameters. The specific support provided by the distributed infrastructure on top of which parameter sweeping applications are run determines the set of operations available to the end user to compose the template task.

In the case of Aneka the parameter sweeping applications are expressed by using the *Parameter Sweeping Model (PSM)* that is modeled on top of the *Task Programming Model*³. The task programming model structures a distributed application as a collection of independent tasks that can be executed in any order. A task is a generic execution unit that can have input and output files, these files are automatically moved in and out of the Aneka cloud when needed.

The Aneka PSM APIs provide the logic for creating the sequence of task instances (jobs) from a template task given the parameters domains. They automatically submit these tasks to the Aneka Cloud and collect back their results that are then presented to the user through the Design Explorer. This particular design, strongly influences the set of operations that are available to the user: for example it is not necessary, as happens in other parameter sweeping models, to specify data movement in the task template but input and output files are automatically moved by Aneka. Moreover, the Aneka PSM APIs provide a set of ready to use commands that can be used to compose the template task of the application. These are:

- *Copy command*: makes a copy of a file on the remote node.
- *Delete command*: deletes a file on the remote node.
- *Execute command*: executes a command on the remote node.
- *Substitute command*: substitutes the occurrences of the parameters with their run time values into a file.
- *Environment command*: sets a collection of environment variables in the shell used to execute the template task on the remote node.

³ For more details about the Task Programming Model please see the Tutorial: *Developing Task Model Applications* in the Aneka distribution or available from the Manjrasoft website.

These commands are specific *GridTask* instances and any other used defined *GridTask* types can be used to define a template task of parameter sweeping applications.

NOTE: The support provided at programming level for Parameter Sweeping Applications is more powerful and advanced than the one provided through the Design Explorer. The template task defining prototype of jobs is modeled as an instance of the *CompositeTask* class that is characterized by a list of *GridTask* instances executed in sequence. Hence, by program any *GridTask* inherited class can be used to compose the task template. By using the Design Explorer it is only possible to compose the tasks by using the five commands listed above, for which a full GUI support has been provided.

4. Design Explorer

The Design Explorer is integrated environment for quickly prototyping Parameter Sweeping applications, controlling and monitoring their execution on Aneka Clouds. It is an environment where user can create, open, and save a project representing their Parameter Sweeping applications. By using a simple step by step wizard users can visually prototype the structure of the template task that will be used to generate all the jobs run on the Aneka Cloud. The template can be saved into a project and run within the environment itself, through which it is possible to monitor its execution and collect some useful statistics.

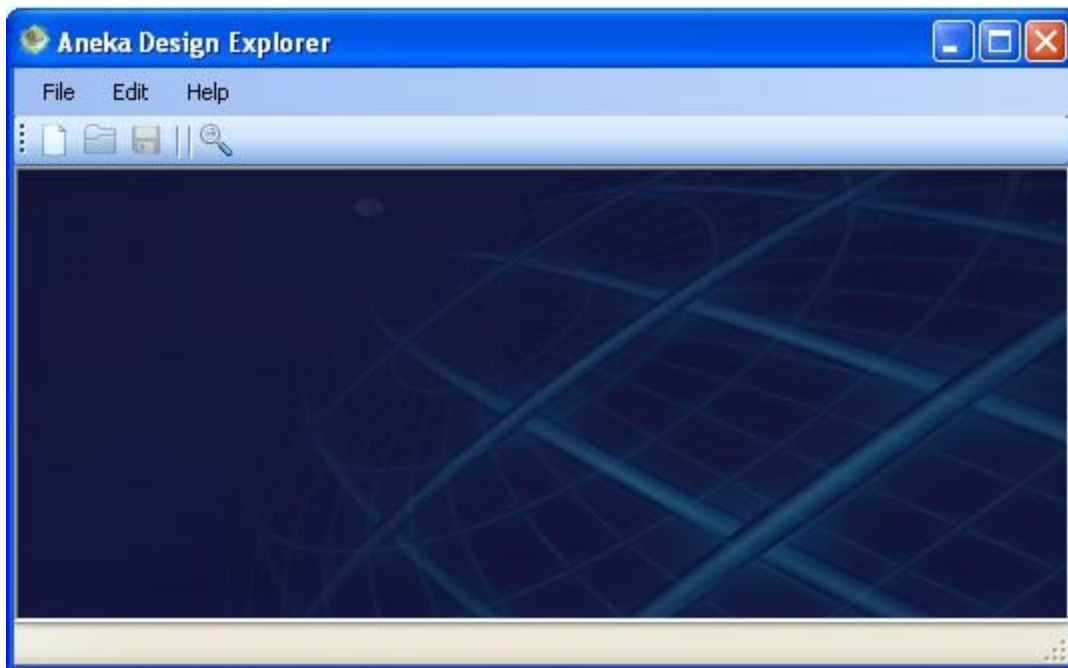


Figure 3. Design Explorer User Interface

The Aneka Design Explorer is located in the bin directory of the Aneka installation

([Programs Folder]\Manjrasoft\[Aneka Version]\bin) and it is accessible from the *Start* → *All Programs* → *Manjrasoft* → *[Aneka Version]* → *Design Explorer* menu item. Figure 3 shows the user interface of the Design Explorer. The menu provides easy access to all the features of the environment:

- *File menu*: provides access to the project wide operations such as create, open, save (and save under a different name) a project.
- *Edit menu*: provides access to the options panel where the user can set the credential information required to access the Aneka Cloud.
- *Help menu*: provides access to this documentation and a brief information dialog box about the Design Explorer itself.

The user interface also features a tool bar that contains the most commonly performed operations (new project, open project, save project, and help). The remaining part of the window constitutes the workspace where the project windows are hosted.

4.1 Creating a Parameter Sweeping Application

In order to create a new Parameter Sweeping application it is necessary to create a new project. This can be done by clicking the leftmost icon in the toolbar representing a blank sheet or selecting the *File* → *New...* menu item.

4.1.1 Application Information

Figure 4 shows the first page of the Aneka Job Wizard that is activated by the previous operation. In this page the user is requested to enter some general details of the application being created such as a name, a description, and the workspace directory.

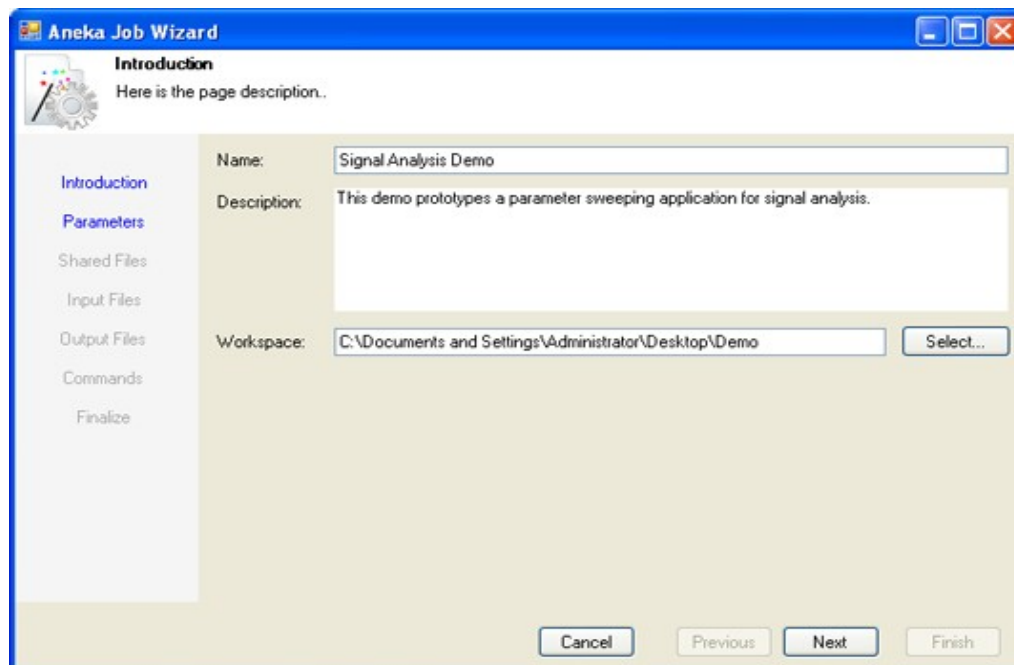


Figure 4. Aneka Job Wizard: Application Details.

On the left side of the wizard it is possible to see all the steps that will be covered in order to define the task template of the Parameter sweeping application. The bottom area of the wizard contains the navigation controls that allow users to move back and forward through the pages of the wizard. It is important to notice that the wizard has an incremental configuration. This means that only the pages that have been successfully validated for what concerns the user input can be accessed and passed over. Once a page is successfully validated or accessed its corresponding name on the left side of the wizard has a blue color and it is possible to directly access to it by clicking on it.

4.1.2 Parameter Definition

Once the user has successfully entered the detail of the application can press the *Next* button and move to the Parameters page where he/she can define all the parameters that control the application.

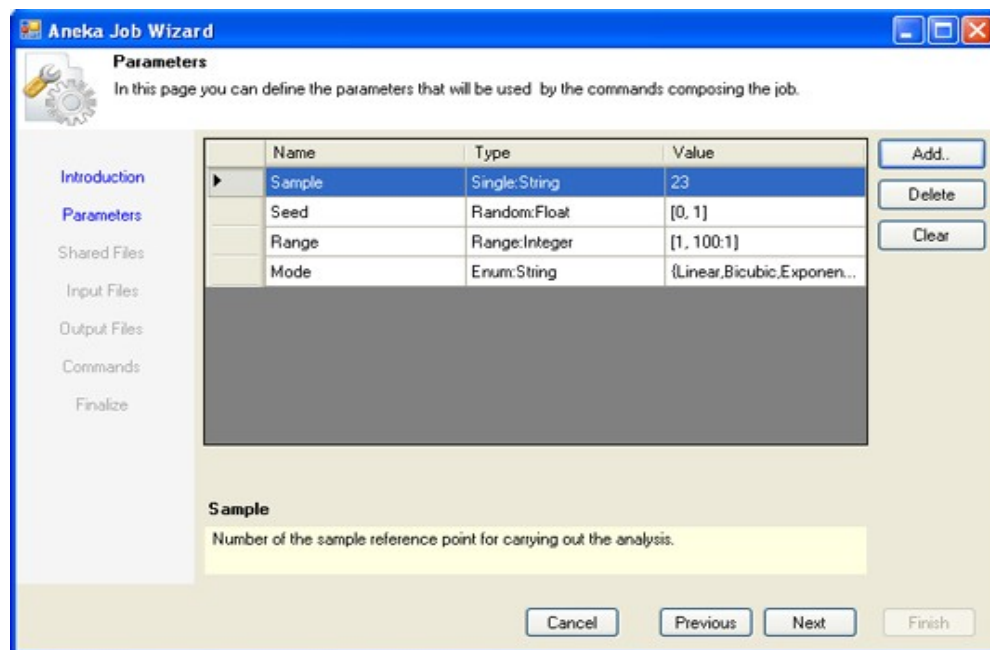


Figure 5. Aneka Job Wizard: Parameter Definition.

Figure 5 shows the Parameters page. It shows the list of parameters currently defined for the application. A parameter is defined by three elements:

- *Name*: represents the name of the parameter that is used to identify it in the task template.
- *Type*: defines the type of parameter.
- *Value*: identifies the value or the values that the parameter can have according to its definition.

By using the *Add*, *Delete*, and *Clear* buttons the user can add a new parameter, delete

the ones currently selected or all the parameters. The Design Explorer allows defining four different types of parameters:

- *Single*: represents a parameter that can assume one single value. The underlying type of the parameter is string.
- *Random*: represents a parameter that can assume a random value within a range limited by a lower and an upper bound. The parameter is a real number.
- *Range*: represents a parameter that can assume a discrete set of values within a limited range and that are generated by starting from the lower bound and adding a step. The parameter is an integer number.
- *Enum*: represents a parameter that can assume a discrete set of values that are defined by the user. The underlying type of the parameter is string.

For all the parameters described above a name is mandatory while the user can enter an additional comment that specifies the role of the parameter.

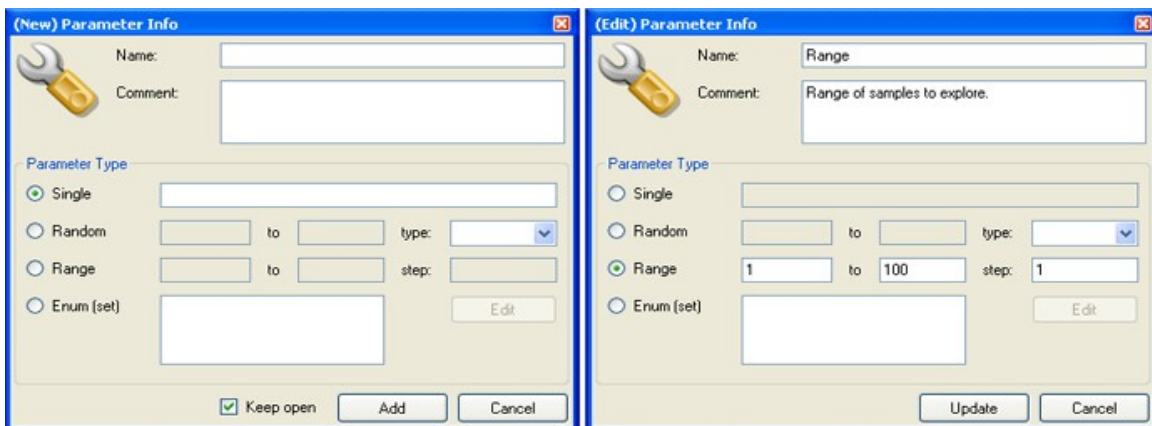


Figure 6. Aneka Job Wizard: New and Edit Parameter Modes.

Figure 6 shows the dialog used to add or edit the properties of a parameter. This dialog shows up if the user presses the *Add* button (*New* mode) or clicks on the row header of one of the existing parameters shown in the list (*Edit* mode). An interesting option is the *Keep open* flag that is visible only in the new mode. This feature, when selected, allows adding more than one parameter by keeping the dialog open after pressing the *Add* button. The *Add* or *Update* buttons also verify that the data entered by the user is valid.

4.1.3 Configuring Shared Files

The template task is generally composed by a sequence of operations. Some of these operations can be the execution of console commands or legacy applications. In this case they are most likely to be the same for all the job instances generated from the template task.

The Design Explorer provides the facility of specifying a collection of files that can of different nature (executable, data files, scripts, etc..) and that are required for

executing every job instance. For example they can represent a database file, the legacy application of an execution command, or something else. These files are automatically transferred by the infrastructure in a transparent manner and made available on the remote node for the job instance.

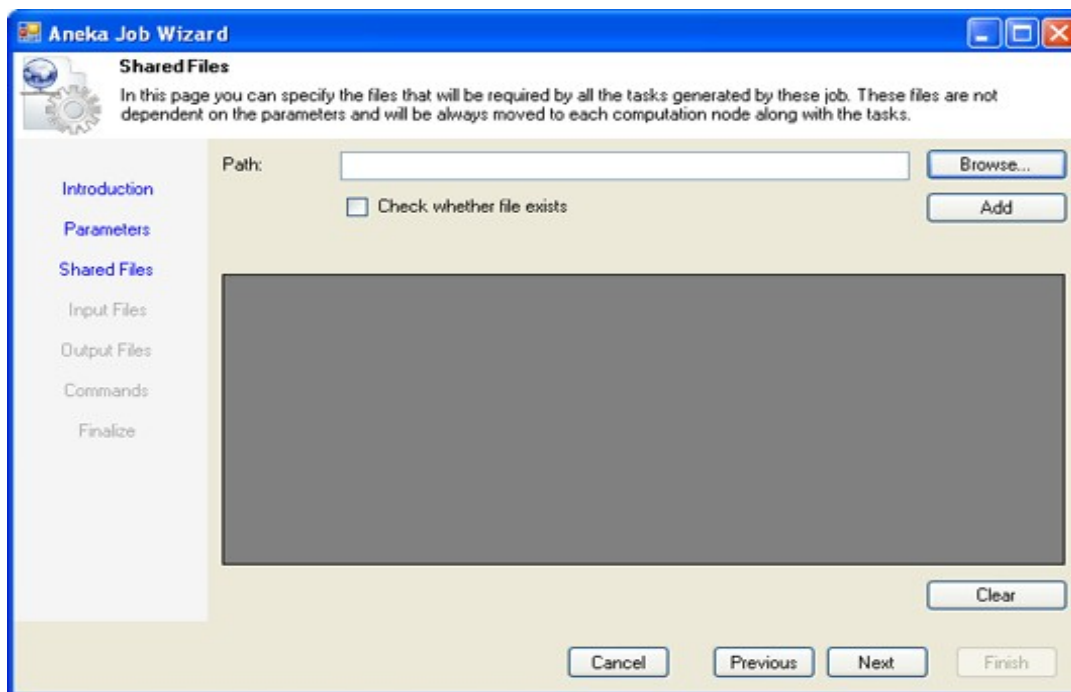


Figure 7. Aneka Job Wizard: Shared Files Page.

Figure 7 shows the Shared Files page. In this page user can select files located in file system reachable from the local machine by pressing the *Browse* button and add them as shared files by pressing *Add*. By clicking the Check whether the file exists flag it is possible to verify the existence of the file in the file system before adding it.

4.1.4 Input and Output Files

The next two steps allow users to specify input and output files for each of the job instances. Differently from the shared files, input and output files can be specialized with parameters. This means that the real name of the file is generated and checked at runtime by the PSM engine.

In order to quickly compose the name of the file (input and output) at the top of the two pages there is a combo box followed by a list that allows users to select the specific parameter that they want to pick in order to compose the file name. Once selected the parameter, the wizard will automatically generate a placeholder for the parameter that will be replaced at runtime by the parameter value. The placeholder takes the form (*\$parameter_name*) and it is inserted at the current location of the cursor in the Input/Output file text box.

It is possible to have three different views for the parameters:

- *All parameters*: shows all the available parameters.

- *User parameters*: shows only the parameters defined by the user in the task template.
- *Special parameters*: shows only the system parameters that are available by default for each job instance. At the moment only the Task Id (Job identifier) parameter is available in this list. Special parameters are characterized by a leading \$ in the parameter name that makes them reserved words.

The first option shows both users and special parameters.

Once the file name is composed it can be added to the list of input/output files by pressing the *Add* button. The file name will be added to the list and checked for its name in order to verify that any parameter placeholder typed by hand is in the correct form. The valid column of the list alerts the user about possibly wrong file names. Once the user has entered all the files, by pressing the *Next* all the files are checked and an error message box is displayed for those that are not valid. Figure 8 shows the Input and Output files pages.

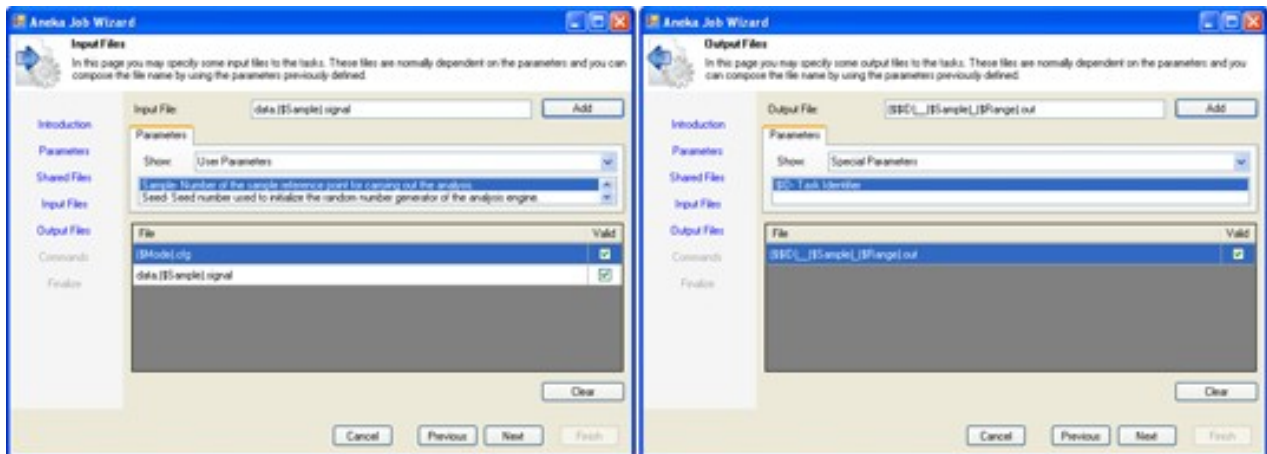


Figure 8. Aneka Job Wizard: Input and Output Files pages.

4.1.5 Task Template Commands

The final step for defining a task template is specifying the sequence of operations that characterize will be executed on the remote node for each of the job instances. This is the last step because the sequence of commands can make use of all the previous elements: parameters, shared, input, and output files.

Figure 9 shows Commands page. The users can select among five different ready to use commands:

- *Copy command (CPY)*: this command completely executes on the remote node and copies a file to another file under a different name but always on the same node. Other implementations of the parameter sweeping model use the copy command to move files from the local client machine to the remote node. With Aneka this task is transparently done and there is no need to do that explicitly in the task template.

- *Delete command (DEL)*: deletes a file on the remote node.
- *Execute command (EXE)*: executes a shell command or console application on the remote node.
- *Substitute command (SUB)*: substitutes the occurrences of the parameters with their run time values into a file.
- *Environment command (ENV)*: sets a collection of environment variables in the shell used to execute the template task on the remote node.

Each of the commands has a specific configuration dialog to compose the command. From these dialogs it is possible to pick up the parameters as explained in the previous sections and the files. All of the dialogs feature an additional tab containing the a list of the files that have been previously entered by the user; they can be filtered by selecting all the files, shared files, input, or output files.

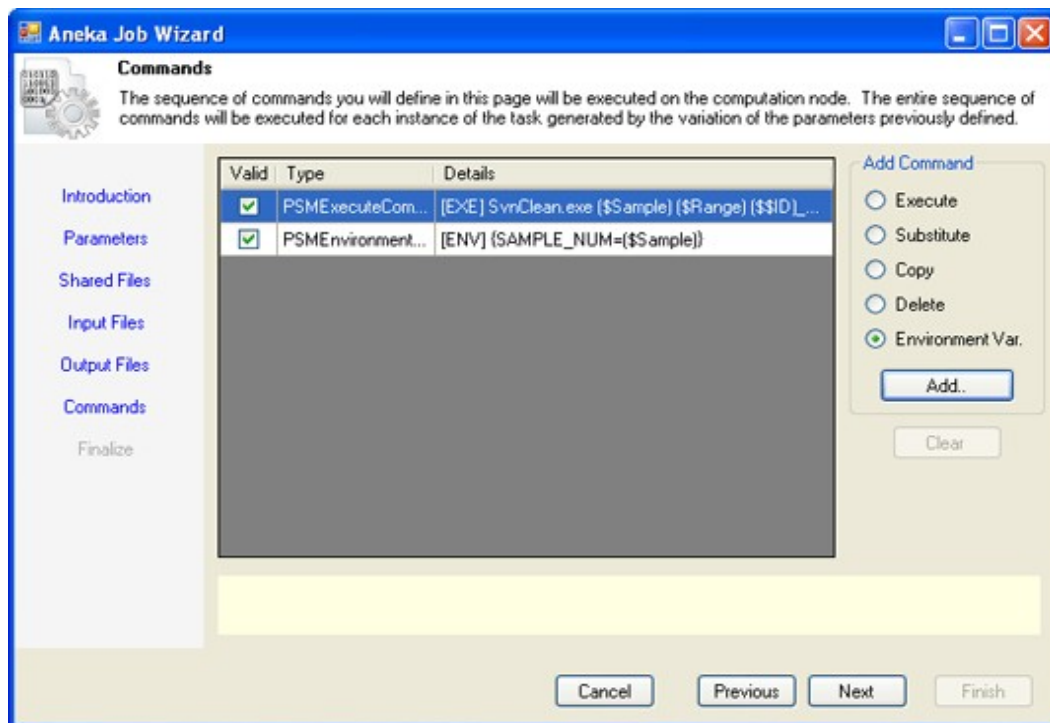


Figure 9. Aneka Job Wizard: Commands Page.

4.1.6 Finalizing the Task Template

The creation of commands is the last step for creating the template. Figure 10 shows the Job Completion page. The user is presented with different options:

- It is possible to save the task template into an XML file. This is accomplished by providing a name into the *Save path* text box or by pressing the *Browse* button to look for an existing file. Once the name is set, it is possible to press the *Save* button.
- It is possible to directly edit the XML source file of the task template. This is

accomplished by clicking the *Edit* button. This feature is only available on the .NET/Windows version; when the code is compiled for the Mono environment an informative message is displayed in place of the XML editor that allows to modify the source of the template.

- It is possible (default action) to open a project and run the parameter sweeping application into the Design Explorer. This option is checked by default and opens up a Project Window through which the users can monitor and execute and modify the template.



Figure 10. Aneka Job Wizard: Job Completion Page.

By pressing the *Finish* button if the *Execute Job on Finish* is checked the project window is open otherwise the entire template goes lost if not saved into an XML file.

4.2 PSM File Structure

The Design Explorer provides a way to serialize into an XML its data. It is possible to save only definition of the task template as a *Parameter Sweeping Model file* (*.psm) or to save the entire project into a file as an *Aneka Parameter Sweep Project file* (*.wbch). The project file is meant to be contain additional data that define the project itself and not only the task template, while the PSM file simply contains the task definition⁴. It is also possible to create a Design Explorer project by starting from a PSM file; in this case the designer will automatically add the missing information and convert the format.

Figure 11 shows the relationship between the two formats. Since this information is subject to change when more features will be added to the Design Explorer we will only concentrate on the description of the PSM file structure that is also the one used

⁴ At the moment the only difference are few enclosing nodes that wrap the content of the PSM file and maintain the name of the project.

by the PSM API exposed by Aneka.

```
<project version="1.0">
  <display-name>Signal analysis demo</display-name>
  <content>
    <psm>
      .....
    </psm>
  </content>
</project>
```

Parameter Sweeping Model file (*.psm)

Aneka Parameter Sweep file (*.wbch)

Figure 11. Aneka Parameter Sweep File and Parameter Sweeping Model File.

The entire structure and content of the PSM file used to illustrate the creation process of the Task Template is depicted in Figure 12. There exist one root node whose tag name is psm that contains the following elements:

- *name node*: contains the name entered in the first step of the wizard for the parameters sweeping application.
- *description node*: contains the description of the parameter sweeping application.
- *workspace node*: contains the path to the workspace for the application execution.
- *parameters node*: contains a collection of nodes that identify all the parameters that have been defined. These nodes all have the *name*, *type*, and *comment* attributes and can be of the following type:
 - *single node*: identifies a single parameter. It contains an additional attribute (*value*) representing the value of the parameter.
 - *range node*: identifies a range parameter. This parameter has three additional attributes that represent the lower bound (*from*), the upper bound (*to*) of the range, and step (*interval*) used to generate numbers.
 - *random node*: identifies a random parameter. This parameter has two additional attribute that represent the lower bound (*minValue*) and the upper bound (*maxValue*) used to define the range from which numbers are randomly picked.
 - *enum node*: identifies an enum parameter. It contains a list of *value* nodes defining the elements of the enumeration set.

```

<psm>
  <name>Signal analysis demo</name>
  <description>...</description>
  <workspace>C:\Documents and Settings\Administrator\Desktop\Demo</workspace>
  <parameters>
    <single name="Sample" type="Single:String" comment="..." value="23" />
    <range name="Range" type="Range:Integer" comment="..." from="0" to="100" interval="1" />
    <random name="Seed" type="Random:Float" comment="..." minValue="0" maxValue="1" />
    <enum name="Mode" type="Enum:String" comment="...">
      <value>Linear</value>
      <value>Bicubic</value>
      <value>Esponential</value>
      <value>Quadratic</value>
    </enum>
  </parameters>
  <sharedFiles>
    <file path="C:\Data\SvnClean.exe" vpath="SvnClean.exe" />
  </sharedFiles>
  <task>
    <inputFiles>
      <file path="($Mode).cfg" vpath="($Mode).cfg" />
      <file path="data.($Sample).signal" vpath="data.($Sample).signal" />
    </inputFiles>
    <outputFiles>
      <file path="($$ID)__$($Sample)_($Range).out" vpath="($$ID)__$($Sample)_($Range).out" />
    </outputFiles>
    <commands>
      <execute cmd="SvnClean.exe" args="($Sample) ($Range) ($$ID)__$($Sample)_($Range).out" />
      <env>
        <variables><variable name="SAMPLE_NUM" value="($Sample)" /></variables>
      </env>
    </commands>
  </task>
</psm>

```

Figure 12. PSM File Content.

- *sharedFiles* node: contains a list of *file* nodes representing the information to locate and safely copy the files from the local file system to the remote node. The *file* node has two attributes *path* and *vpath*. The first contains the local path of the file, while the second identifies the path on the remote node that is merely represented by the file name of the file.
- *task* node: contains the definition of the input and output files and the operations that have to be performed on the remote node for each of the job instances. This node has three major nodes:
 - *inputs* node: contains a list of *file* nodes representing the input files.
 - *outputs* node: contains a list of *file* nodes representing the output files.
 - *commands* node: the possible nodes contained in this node are the following:
 - *copy* node: stores the information related to the copy command. It contains two attributes *src* and *dest* that respectively represent the

source path to the file and the target path to copy.

- *delete node*: stores the information related to the delete command. It contains only one attribute *file* representing the path to the file to delete.
- *execute node*: stores the information related to execute command. It contains two attributes *cmd* and *args* that respectively represent the command to execute and its arguments.
- *substitute node*: stores the information related to the substitute command. It contains two attributes *src* and *dest* that respectively represent the original path to the file and the path to the new file with the occurrences of parameters replaced with the corresponding values.
- *env node*: stores the information about the environment variables. It contains a *variables* node featuring a list of *variable* node whose attributes name and value respectively identify the name and the value of the environment variable to set.

By editing directly the PSM file it is possible to change the content of the task template without the support of the Design Explorer. This opportunity can be exploited by other applications that as a result of their execution can produce PSM file that can be used in the Design Explorer.

4.3 Managing and Executing Parameter Sweeping Applications

Once the user has created the task template the Design Explorer will open a project window that allows modifying the template and running the corresponding parameter sweeping application.

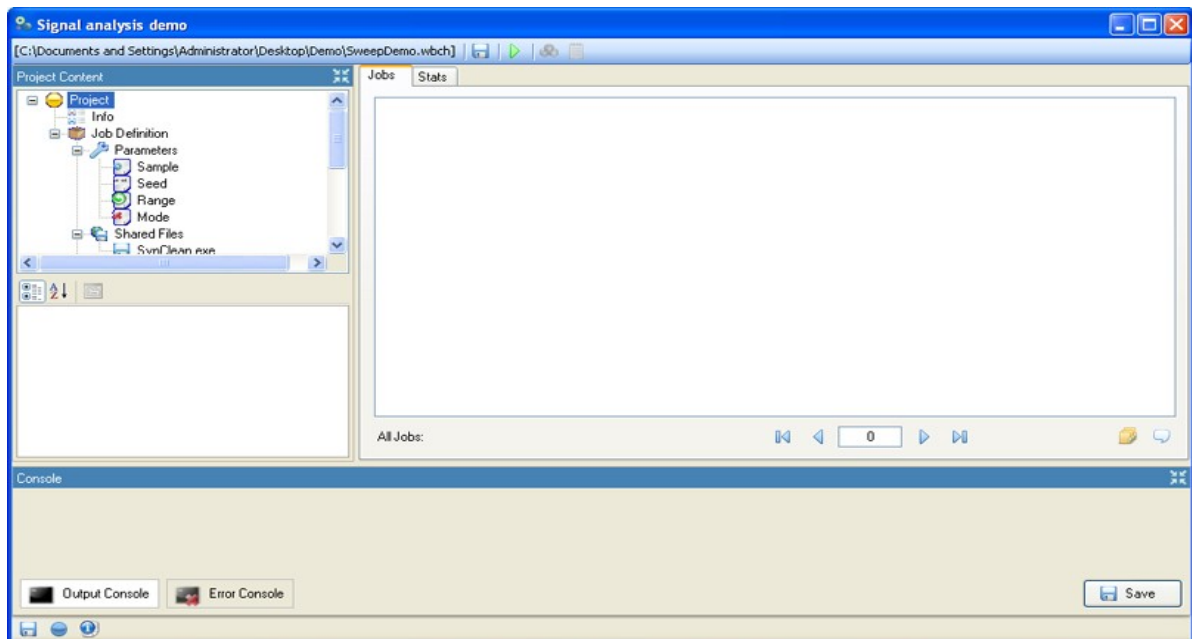


Figure 13. Project Window.


4.3.1 Project Window Layout



Figure 13 shows the project window. The same window is obtained if the user selects *File* → *Open...* and chooses a .wbch or .psm file. There are three main areas in the project window that is worth noticing:

- *left pane*: the left pane features a tree view where the users can see and modify the definition of the task template. By clicking on the nodes of the tree at the bottom of the pane it is possible to see the properties of each node and modify those that are not read only.
- *right pane*: the right pane is composed by two tabs and hosts a dynamic view of the parameter sweeping application while it is running on the Cloud. More precisely, the *Jobs* tab hosts the list of jobs generated by the parameter sweeping application, the *Stats* tab collects some statistics about the whole application and estimates the completion time.
- *bottom panes*: the bottom panes contain two consoles. The *Output console* contains the log of the PSM engine while the *Error console* dumps all the errors occurred while interacting with Aneka. These two console are only active while the application is running and the user can save their content by pressing the *Save* button.

The project window also contains a toolbar that shows the location of the project file and some buttons to save the project, control its execution, and control the appearance of the window. At the bottom of the window a statusbar contains the some additional information about the project such as:

- *Save status*: a floppy disk indicates whether the project has been saved since the last changes have been applied. If the icon representing the floppy disk is blue the project has to be saved, if the icon is black the project has been saved.
- *Running status*: a small ball icon indicates whether the project is running or not. A blue color indicates that the application is idle and has not been run yet. A green or yellow color indicates a running application: if the color is yellow some error occurred, while a green color stands for a flawless run. A red color indicates a permanent failure.
- *Project information*: the last icon in the statusbar provides access to some information about the project itself. At the moment the information displayed when clicking the icon are limited to the name of the project and the location of the corresponding project file.
- *Contextual information*: the portion of the statusbar following the icons is used by the Design Explorer to provide information about the last action performed. The user can quickly have a look at this area to know what was the last task performed by the environment for that project.

The layout of the project window can be changed by hiding some of the panels that compose it. In particular the bottom panes and the left pane can be hidden by clicking the collapse icon  at the top right edge of the pane header. When the left pane is

hidden the icon  in the toolbar becomes active and by clicking it it is possible to restore the pane. The same applies for the bottom pane that is controlled by the icon  in the toolbar.

4.3.2 Editing the Task Template

If the project is not running the user can still edit the parameters of the task template. As shown before, the project window provides users with a tree view where it is possible to browse the structure of the task template. The structure of the tree is similar to the one of the XML file that stores the information of the template. Hence users can intuitively look for the elements they want to change.

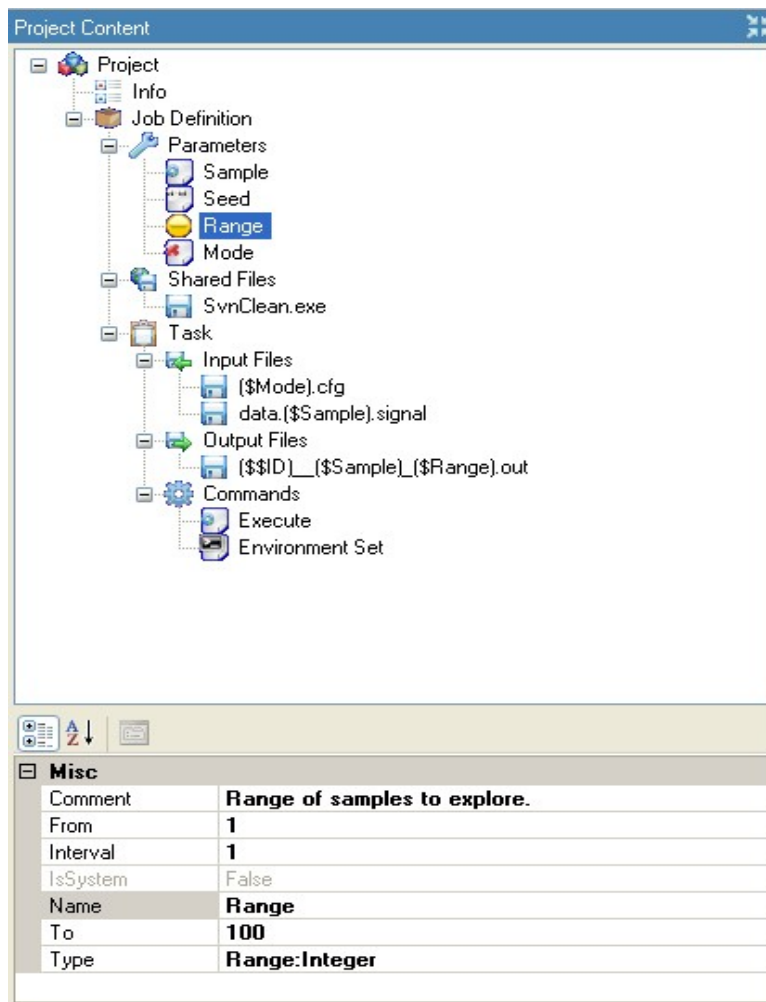


Figure 14. Project Content: Task Template Structure.

Figure 14 shows the content of the left pane of the project taken as case study. There are two main areas: the top area that shows the structure of the task template and the bottom area providing contextual information about the tree node that is selected in the top area. All the properties in the bottom area that are showed in bold can be changed, those who are grayed not.

The content of the task template can only be changed if and only if the project is not

running.

NOTE: The capability of changing the configuration of the task template from the project window is limited and has to be performed with care. Differently from the wizard the left pane does not perform all the checks against the values entered by the user and it is very easy to make mistakes that compromise the execution of the application. For example, while editing the elements that involve parameter placeholders there is no check to ensure that the new value entered by the user is legally valid and this will make the application not run properly. For this reason, this feature has to be used very carefully and to for example change the name of files, the bound values of range parameters or random parameters, or the value the value of single and enum parameters

4.3.3 Connecting to Aneka

In order to execute a project it is necessary to authenticate against the Aneka Cloud. The user has to provide the access point to the cloud and valid user credentials. This can be done by selecting *E*dit → *P*references... . Figure 15 shows the dialog where the user can customize the connection to Aneka.



Figure 15. Aneka Connection Form.

In order to connect to Aneka the Design Explorer need to know the access point to the Cloud. This is a simple address composed by three components:

- *Internet address*: an internet address (IP or DNS name) representing the address of the access point of the Aneka Cloud. If the access point to the Cloud is installed on the local machine the user can enter *localhost*.
- *Connection port*: the port number where the access point to the Cloud is listening for connections. By default Aneka listens on the 9090 port but during setup this information can be changed. Hence, it is important that user know what is the correct port number.
- *Service name*: the name of the service that is exposed by the access point to the Cloud. This is always *Aneka* and must not be changed.



The rest of the information required are the user credentials that are constituted by

the user name and the password of a valid Aneka user. This information must be known in order to access the Cloud.

Once the user has entered valid information he or she can click the OK button to save them into the Design Explorer.

NOTE: The information about the connection to Aneka are **not** per project but are a property of the Design Explorer. This means that if the user is running multiple projects at the same time they will run on the same Aneka Cloud under the same user. It is possible to start one project, change the connection details, and then start another project in order to have a per project setting but this practice is not considered safe.

4.3.4 Running the Project

Once the user has opened or created a new project he or she can run it by clicking the play icon  in the project window toolbar. As long as the project is running the run icon shows the stop symbol  and by clicking on it is possible to terminate the execution of the project. Once the project naturally terminates or it is stopped the Design Explorer will automatically restore the play icon.

Once the user runs the project and there are no problems in connecting with the Aneka Cloud the Parameter Sweeping application starts and the two tabs on the right pane are filled with information about the running application. In particular, the application will generate all the jobs from the template task and the PSM engine will queue them to the Aneka Cloud for their execution. At this point the *Jobs* tab will contain the list of all the jobs of the application while the *Stats* tab will feature a pie chart together with some time statistics about the medium time spent by the job in each status allowed by the system.

From what concerns the execution itself a subdirectory in the workspace folder specified in the project will be created in order to store all the output files of the current application execution. The name of this directory is composed as follows:

PSM - Project Name_GUID

Where *GUID* is a Globally unique identifier automatically generated by the PSM engine and ensured to be unique in the world. Each time the same project is run a new folder with a new value of the GUID component is created in the workspace of the project. This avoids that the results of different runs clash in the same folder.

While the project is running most of the activity is concentrated in the right pane of the project window that hosts two tabs: Jobs Tab and Stats Tab. The bottom part of the project window is occupied by two consoles that simply track the messages and the errors generated by the system. In the next three sections we will illustrate the features of these components.

4.3.5 Jobs Visualization

The Jobs Tab is the main view of the application running. It gives you a complete view of all the jobs generated by the application and shows their current status.

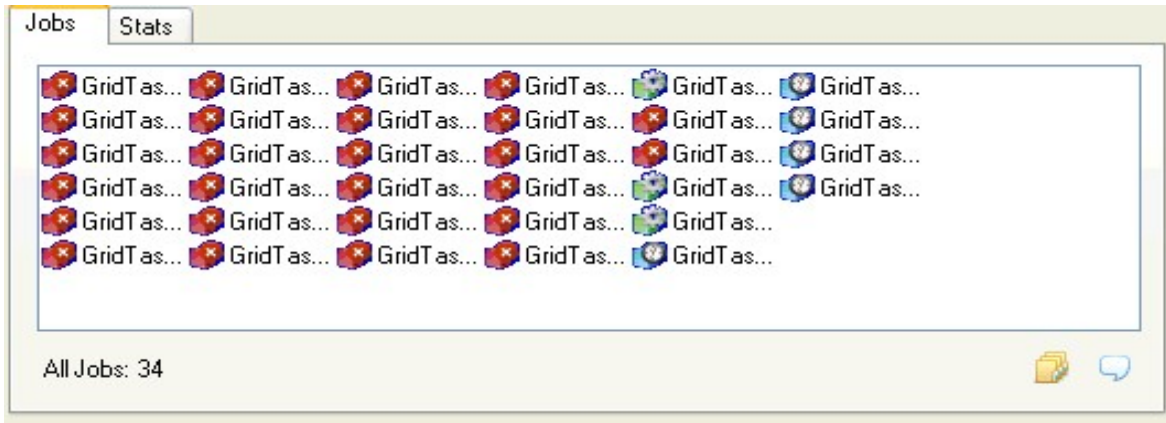


Figure 16. Design Explorer Jobs Tab.

Figure 16 shows the content of the Jobs tab while the application is running. The jobs are identified by an icon and a friendly name that is automatically generated by the PSM engine. The icon identifies the status of the job. Here is the list of the different states in which a job can be:



Unsubmitted

The job has not been submitted to the Aneka Cloud yet. When the Parameter Sweeping application starts all the jobs are in this state.



StagingIn

The PSM engine is uploading input files to the Aneka Cloud. This state appears if and only if the task template has some input files defined.



Queued

The job has been submitted to the Aneka Cloud and has been put into the scheduler queue. At this stage the job is not running but it is waiting to be dispatched to a resource for executing.



Running

The job has been dispatched to some node and it is running. The Aneka Cloud is then waiting for its completion to collect the results and send them back to the scheduler.



StagingOut

The job has completed successfully its execution and the results have been collected and put into the Aneka Cloud storage. The

PSM engine is downloading the output files of the job to the local machine. This state only appears if the task template defines some output file.



Completed

The job is completed and all the output files, if any, have been downloaded to the local machine. This state identifies a successful completion.



Failed

The job is failed. This is state can imply different things: the job execution has failed or there has been an error while moving the job to or within the Aneka Cloud that caused its failure.



Aborted

The job has been aborted by the user. This generally happens when the user stops the execution of a specific job or stops the execution of the entire application. The job can fall into this state even if staging in of files fails.

The jobs tab also features a set of controls that information that can simplify the management of large number of jobs. In this case there are only 34 jobs generated and they can fit within the same window. In case the number of jobs is huge an additional navigation control shows up at the bottom of the Jobs tabs that helps the user to navigate between the pages into which the collection of jobs is divided.

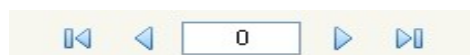


Figure 17. Jobs Navigation Control.

The control allows user to quickly move to the first and the last page and to browse the pages by moving back and forward or simply entering the number of the page to view.

The bottom area of the Jobs tab is completed by an informative text containing the number of all the jobs generated on the left and two icons on the right side. These two icons provide some information about the status of the Job Manager) 📁 and the visualization settings of the Jobs tab (such as the current view mode, the total number of jobs, and the number of jobs per page) 🗨️. The Jobs tab can also filter the list of Jobs according to their state in order to show only the interesting information for the user. In order to do so it is possible to right click with the mouse on the white area containing the Jobs and select the *Filter* item from the context menu that appears.

Figure 18 shows a possible configuration of the context menu. In the figure all the possible states are selected. The user can individually select the states that want to browse or click on the All or None options that respectively show all the jobs or none of them. The Jobs Tab also allows to select a different visualization mode and Figure

19 shows the possible options under the View submenu of the same context menu.

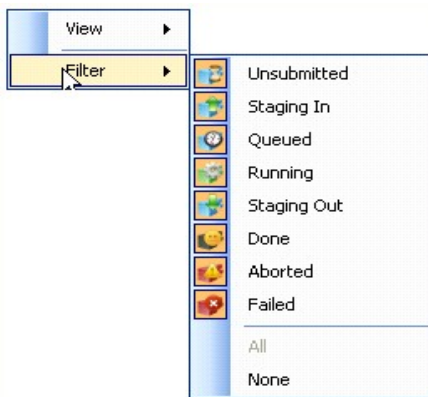


Figure 18. State Filter Context Menu.

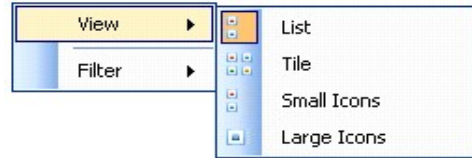


Figure 19. List View Mode Menu.

It is possible to use a simple list, a tiled view, a small or a large icons view. The last two options organize the jobs into classes that map to their state and provide a classified view of all the jobs.

4.3.6 Statistical Data

The Stats tab provides users with a statistical view of the application. In particular it collects the statistics of the execution and shows an overall view of the job state distribution by means of a pie chart.

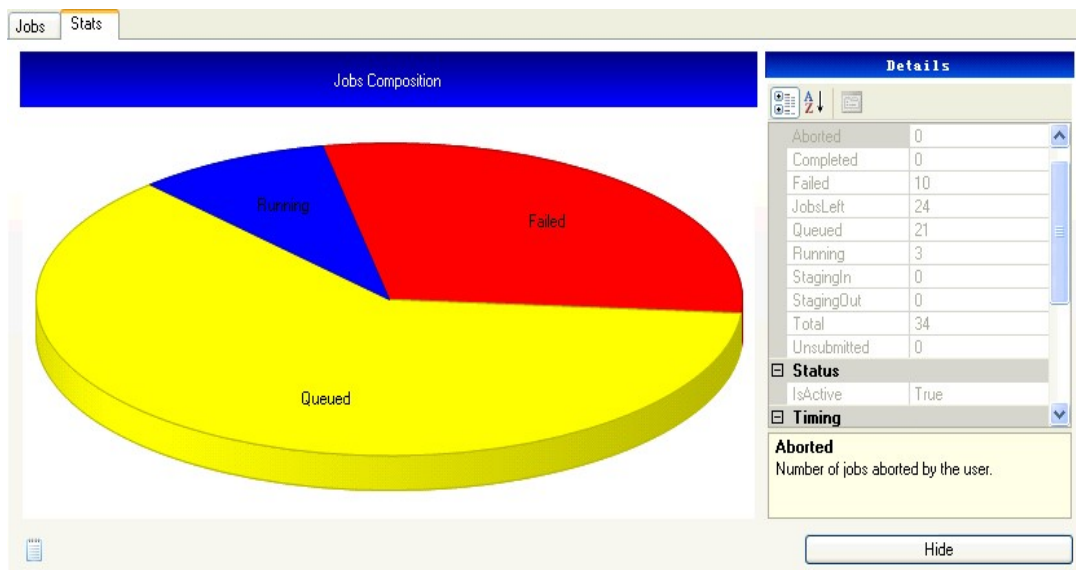




Figure 20. Stats Tab.

Figure 20 shows the content of the *Stats* tab while the project is running. As it can be noticed the tab is divided into three major areas:

- *Pie chart*: the chart shows the jobs composition while the application is running. As long as new jobs change their state the pie chart is updated. Each state is shown in a different color and all the jobs with the same state are grouped together in the same pie slice.
- *Details panel*: this panel gives the break down of the jobs composition by showing for each state how many jobs are in that state and the total number of jobs. The panel also provides some very basic time estimates and the elapsed time since the application started. Users can also control the collection interval of the data, the default value is set to 3000 ms but according to the nature of the jobs a longer or shorter interval can provide a better refresh.
- *Bottom area*: the bottom area contains two elements a control button that is used to show or hide the details panel and an icon  that allows users to show the legenda for the pie chart. This icon is only active while the application is running.

Except for showing and hiding the details pane this tab does not provide any interesting interaction but it is useful for having a global view and a detailed view of the application.

Figure 19 shows the dialog that pops up when the user clicks on the  icon at the bottom of the *Stats* tab. The dialog contains a legenda that maps the colors used in the pie chart with the state values of the jobs. The state names are written in each of the slices that are currently composing the pie chart. By using the legenda it is possible to see all the available colors even those who are not present in the pie chart because there is no job in that state.

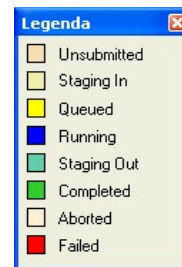


Figure 21. Pie Chart Legenda.

The legenda acts as a modal dialog. This means that the entire user interface is blocked until the legenda is open. In order to interact with any other control in the GUI of the Design Explorer it is necessary to close the legenda.

4.3.7 Analyzing the Console

The bottom pane provides user with access to two console: *Output Console* and *Error Console*.

The *Output Console* is used to log the interaction of the Design Explorer with the Aneka Cloud while running the application, while the *Error Console* is used to trace all the errors that occur during the interaction. The user can switch between the two console by clicking the corresponding buttons at the bottom of the pane. The *Output Console* logs the interaction of the Design Explorer with Aneka. In particular what is interesting is the tracking the status change of the different jobs while they are executed in the Aneka Cloud. Figure 22 shows the an example of the content of the *Output Console*. While the Jobs and Stats tabs provide a visual information about the execution of the application, the console shows detailed text information about state

transitions and the identifier of the nodes where each of the job is executed. This information can then be saved to file by pressing the *Save* button, while the *Clear* button can be used to clean the console.



Figure 22. Design Explorer Output Console.

The *Error Console* mostly traces exception occurred during execution and error messages. The information displayed about exceptions are the following:

- *Exception type*: the .NET type of the exception occurred at program level.
- *Exception message*: informative message describing the nature of the exception.
- *Stack trace*: the exact point in the execution stack where the exception has occurred.

This information, except for the exception message, are not of a great help but can be used to provide an helpful feedback to the Aneka development team for dealing with the problem.

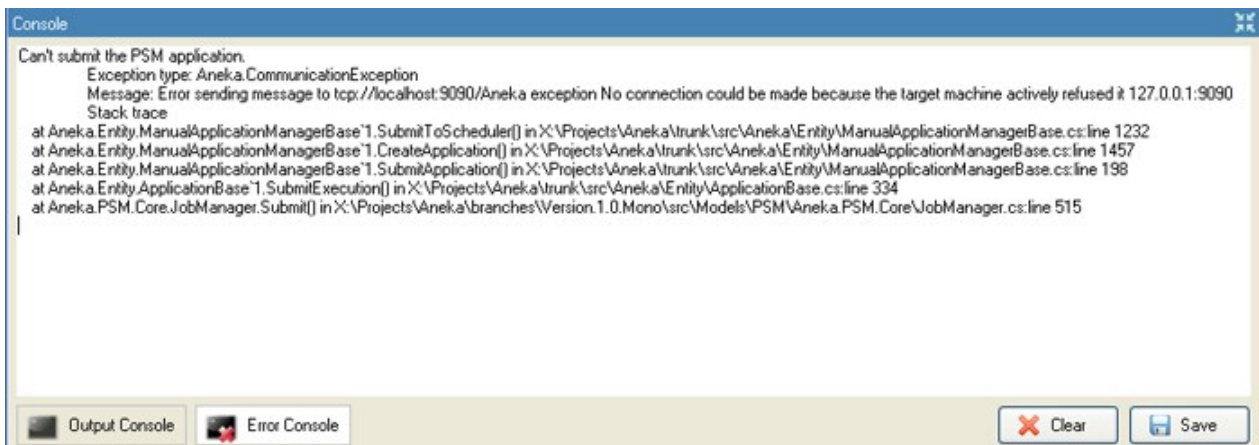


Figure 23. Design Explorer Error Console.

Figure 23 shows a possible content of the *Error Console*. As an example the error

occurring if the user does not provide a valid connection information is reported. As happens for the Output Console the user can save the content of the Error Console to a file by pressing the *Save* button.

5. Example

In this section we will guide the user to create a simple parameter sweeping application that can be used to demonstrate the features discussed so far of the Design Explorer. In order to simplify the example we will use a ready to use application from the biology field that is called BLAST.

5.1 BLAST

BLAST (Basic Local Alignment Search Tool) is a tools for looking for similarities between a given sequence of genes and those stored into classified databases. The BLAST application is available for download from the National Centre for Biotechnology Information (NCBI)⁵ website that also provides a classified repository of all the databases that can be used to search for similarities. The role of these databases is really important since it is one of the most important knowledge repository for genome sequences and helps researchers to identify and study sequences of genes.

5.1.1 BLAST Distribution

BLAST is a set of tools that allows performing advanced queries against genome databases and it is available for different platforms (Windows, Linux, Mac OS X, ...). In this simple example we will concentrate our attention only on one component that is *blastall*, which performs the basic search for a genome sequence into a given database.

5.1.2 Executing a BLAST Query

In order to search a given database it is necessary to prepare (format) it properly. Another tool in the BLAST distribution allows formatting the database in a way in which the searches made by *blastall* are possible: *formatdb*. This tool takes as input a database file and create a set of file indexes that are used by the *blastall* application. Once these indexes are created the original database file is not required anymore.

The search of a specific sequence of genes against a given database is then performed by the following steps:

- download the BLAST distribution (*blastall*, *formatdb*) from the NCBI website
- download the *<database>* file from the NCBI website
- execute: *formatdb -i <database> -p F -o T*
- execute: *blastall -p blastn -d <database> -i <sequence> -o <result>*

5 www.ncbi.nlm.nih.gov/BLAST/

where:

- <database> is the database file name downloaded from the NCBI website
- <sequence> is the file name of the sequence of genes to look for
- <result> is file name where the result of the search are stored

The content of the <sequence> file is basically a sequence of characters representing the genes we are looking for. An example of the content of this file is the following:

>Test

```
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAGAGTGTCTGATAGCAGC
TTCTGAACTGGTTACCTGCCGTGAGTAAATTTAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAA
TATAGGCATAGCGCACAGACAGATAAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACC
ATTACCACCACCATCACCATTACCACAGGTAACGGTGCGGGCTGACGCGTACAGGAAACACAGAAAAAAG
CCCGCACCTGACAGTGCGGGCTTTTTTTTTTCGACCAAAGGTAACGAGGTAACAACCATGCGAGTGTGAA
GTTTCGGCGGTACATCAGTGGCAAATGCAGAACGTTTTCTGCGTGTTGCCGATATTCTGGAAAGCAATGCC
AGGCAGGGGCAGGTGGCCACCGTCCTCTCTGCCCCCGCCAAAATCACCAACCACCTGGTGGCGATGATTG
AAAAAACCATTAGCGGCCAGGATGCTTTACCCAATATCAGCGATGCCGAACGTATTTTTTGCCGAACTTTT
```

The content of the <result> file is a list of hits in the selected database.

5.1.3 Parallelizing a BLAST Task

There are many way to parallelize a BLAST query against a database. In this simple example we will use the Parameter Sweeping Model in order to automatically perform multiple BLAST queries against the same database over a distributed infrastructure.

Since the database is the same it can be initially prepared for the search by performing the *formatdb* operation offline. Hence, the only operation that will be distributed will be the *blastall* command that will cover different sequence files.

An alternative approach that can be taken is to perform the search operation against multiple databases. In this case the since the database changes it becomes a parameter of the application and the *formatdb* operation has to be done as an execute task of the job that is distributed.

Within the context of this example we will only consider the first approach and we will use as search database the one containing the gene sequences of the *Escheria Coli* (*ecoli.nt*) available from the NCBI website.

For convenience all the files required to run the example are provided in the following folder:

[Program Files]\Manjrasoft\[Aneka Version]\examples\Tutorial\Parameter Sweeping Model\BLAST

This folder contains the following files:

- *ecoli.nt*: database of gene sequences of the Escheria Coli

- *blastall.exe*: Windows version of the blastall program.
- *formatdb.exe*: Windows version of the formatdb program.
- *seq0.txt*, ..., *seq2.txt*: sequence input files.
- *blast.psm*: Parameter Sweeping Model file for the blast task template.
- *blast.wbch*: Aneka Parameter Sweep file for the blast project.

The last two files are provided for convenience and will be created by following the steps that are provided with this example.

5.2 Creating the Parameter Sweeping Application for BLAST

5.2.1 Identifying Parameters

The first step to do while creating a task template is to identify the parameters that are involved in the application and their nature. In this case since we have decided to perform multiple searches against the same database we will have the following parameters:

- *SequenceFile*: it could be a range or an enum parameter.
- *DatabaseFile*: it is a fixed parameter the value is *ecoli.nt*.
- *ResultFile*: it is a parameter depending on the *SequenceFile* parameter.

We can collapse the *SequenceFile* and *ResultFile* parameters into a single parameter *SeqNum* of type *Range:Integer [0,2:1]* and compose the names of the sequence and the result files accordingly.

5.2.2 Selecting Shared Files

In order to perform the BLAST search we need to prepare the database first by executing the command:

```
formatdb -i ecoli.nt -p F -o T
```

This operation will create the following index files that are required by the *blastall* program to perform the search:

- *ecoli.nt.nhr*
- *ecoli.nt.nin*
- *ecoli.nt.nnd*
- *ecoli.nt.nni*
- *ecoli.nt.nsd*
- *ecoli.nt.nsi*

These files together with the *blastall.exe* executable are the shared files of out

application since they are required by each of the jobs that are created from the task template.

5.2.3 Identifying Input and Output Files

The *blastall* command requires the set of indexes to perform the search, the specific sequence file to be looked for and it produces the a result files containing the matches found in the database.

Since all the index files are already provided to each job (as well as the *blastall.exe* executable) the only input file to define in the task template is the sequence file. For what concerns the output files we have only one file that is represented by the result file produced by the *blastall* command. Both input and output files are dependent on the *SeqNum* parameter that can be used to compose their names as follows:

- Input file: *[path to]seq(\$SeqNum).txt*⁶
- Output file: *result(\$SeqNum).txt*

There are no other files to consider.

5.2.4 Creating the Task Commands

The commands section of the Task Template will only contain one single command that is an execution command (EXE) that will run the *blastall* program. In this case we will have the following settings:

- cmd: *blastall.exe*
- args: *-p blastn -d (\$Database) -i seq(\$SeqNum).txt -o result(\$SeqNum).txt*

This command will execute the BLAST search and produce the result file.

5.2.5 Using the Wizard and Creating the .psm and .wbch Files

The table below shows a summary of the data that need to be entered in the wizard in order to create the task template:

Parameters	Database (Single: [path to] <i>ecoli.nt</i>) SeqNum (Range:Integer [0,2:1])
Shared Files	[path to] <i>blastall.exe</i> [path to] <i>ecoli.nt.nhr</i> [path to] <i>ecoli.nt.nin</i> [path to] <i>ecoli.nt.nsd</i> [path to] <i>ecoli.nt.nnd</i> [path to] <i>ecoli.nt.nni</i> [path to] <i>ecoli.nt.nsi</i>
Input Files	[path to] <i>seq(\$SeqNum).txt</i>

⁶ [Path to] identifies the full path to the *seqK.txt* files where K = 0,1,2.

Output Files	result(\$SeqNum).txt
Commands	[EXE] cmd: blastall.exe args: -p blastn -d (\$Database) -i seq(\$SeqNum).txt -o result(\$SeqNum).txt

In order to create the task template it is only necessary to select *File* → *New...* and provide a name, a description, and a workspace directory for the project. Figure 24 shows the first page of the wizard where all this information is entered.

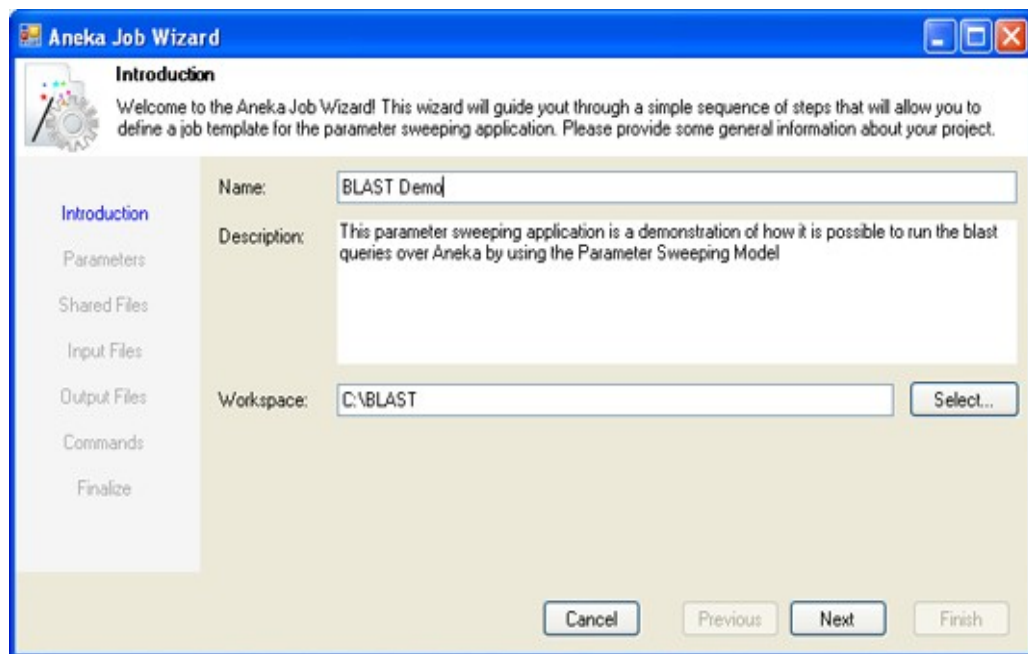


Figure 24. Entering the Details of the BLAST Project.

The next steps of the wizard will add the information in the table. Once the user has reached the *Finalize* page he or she can save the template with the *blast.psm* file name and press the *Save* button as shown in Figure 25. At this point the Designer Explorer saves the Parameter Sweeping Model file, creates an Aneka Parameter Sweep project, and loads the project window ready to be executed on the Aneka Cloud.

The first thing to do is to save the project. By using the wizard the user has selected the option to save the Parameter Sweeping Model file and this does not automatically save the project file too. In order to save the project it is sufficient to select *File* → *Save* and since the project is not saved a Save File dialog will pop up and the user will be asked to provide a name for the project. We provide the *blast.wbch* file name and save the project file into the Workspace directory of the project.



Figure 25. Saving the BLAST Task Template.

Figure 26 and 27 show the content of the *blast.psm* file and the *blast.wbch* file. The user can check whether, except for some directory information, the structure and the content of the file generated while trying the example are the same.

```
<?xml version="1.0" encoding="Windows-1252"?>
<psm>
  <name>BLAST Demo</name>
  <description>...</description>
  <workspace>C:\BLAST</workspace>
  <parameters>
    <range name="SeqNumber" type="Range:Integer" comment="..." from="0" to="2" interval="1" />
    <single name="Database" type="Single:String" comment="..." value="ecoli.nt" />
  </parameters>
  <sharedFiles>
    <file path="...\BLAST\blastall.exe" vpath="blastall.exe" />
    <file path="...\BLAST\ecoli.nt.nhr" vpath="ecoli.nt.nhr" />
    <file path="...\BLAST\ecoli.nt.nin" vpath="ecoli.nt.nin" />
    <file path="...\BLAST\ecoli.nt.nnd" vpath="ecoli.nt.nnd" />
    <file path="...\BLAST\ecoli.nt.nni" vpath="ecoli.nt.nni" />
    <file path="...\BLAST\ecoli.nt.nsd" vpath="ecoli.nt.nsd" />
    <file path="...\BLAST\ecoli.nt.nsi" vpath="ecoli.nt.nsi" />
    <file path="...\BLAST\ecoli.nt.nsq" vpath="ecoli.nt.nsq" />
  </sharedFiles>
  <task>
    <inputs>
      <file path="seq($SeqNumber).txt" vpath="seq($SeqNumber).txt" />
    </inputs>
    <outputs>
      <file path="result($SeqNumber).txt" vpath="result($SeqNumber).txt" />
    </outputs>
    <commands>
      <execute cmd="blastall.exe" args="-p blastn -d ($Database) -i seq($SeqNumber).txt -o result($SeqNumber).txt" />
    </commands>
  </task>
</psm>
```

Figure 26. BLAST PSM File Content.

```

<?xml version="1.0" encoding="Windows-1252"?>
<psm>
  <name>BLAST Demo</name>
  <description>...</description>
  <workspace>C:\BLAST</workspace>
  <parameters>
    <range name="SeqNumber" type="Range:Integer" comment="..." from="0" to="2" interval="1" />
    <single name="Database" type="Single:String" comment="..." value="ecoli.nt" />
  </parameters>
  <sharedFiles>
    <file path="...\BLAST\blastall.exe" vpath="blastall.exe" />
    <file path="...\BLAST\ecoli.nt.nhr" vpath="ecoli.nt.nhr" />
    <file path="...\BLAST\ecoli.nt.nin" vpath="ecoli.nt.nin" />
    <file path="...\BLAST\ecoli.nt.nnd" vpath="ecoli.nt.nnd" />
    <file path="...\BLAST\ecoli.nt.nni" vpath="ecoli.nt.nni" />
    <file path="...\BLAST\ecoli.nt.nsd" vpath="ecoli.nt.nsd" />
    <file path="...\BLAST\ecoli.nt.nsi" vpath="ecoli.nt.nsi" />
    <file path="...\BLAST\ecoli.nt.nsq" vpath="ecoli.nt.nsq" />
  </sharedFiles>
  <task>
    <inputs>
      <file path="seq($SeqNumber).txt" vpath="seq($SeqNumber).txt" />
    </inputs>
    <outputs>
      <file path="result($SeqNumber).txt" vpath="result($SeqNumber).txt" />
    </outputs>
    <commands>
      <execute cmd="blastall.exe" args="-p blastn -d ($Database) -i seq($SeqNumber).txt -o result($SeqNumber).txt" />
    </commands>
  </task>
</psm>

```

Figure 27. BLAST WBCH File Content.

NOTE: In the two listings some long information about the path to the files and the description of the parameters have been substituted with three dots in order to properly display the content of the files.

At this stage all the data of the project is saved and the application is ready to be run.

5.3 Running the BLAST Project

In order to run the project it is necessary to properly set up the connection information to the Aneka Cloud. This information as already shown can be provided by selecting the menu voice: *Edit* → *Preferences...* This command pops up the form shown in Figure 15 (see Page 19). This is the form where the user has to enter user name, password, and the address to a Aneka Cloud access point as described in section 4.3.3.

Once the connection details are entered the user can run the application by clicking on the play icon of the toolbar and the PSM engine will start submitting jobs to the Aneka Cloud. The current setup of the task template will only generate three jobs: one for each sequence file that has to be searched.

Figure 28 shows the BLAST Demo application running. Once the application is finished the user can find the result files into the PSM - BLAST Demo_<GUID> subdirectory in the project workspace folder.

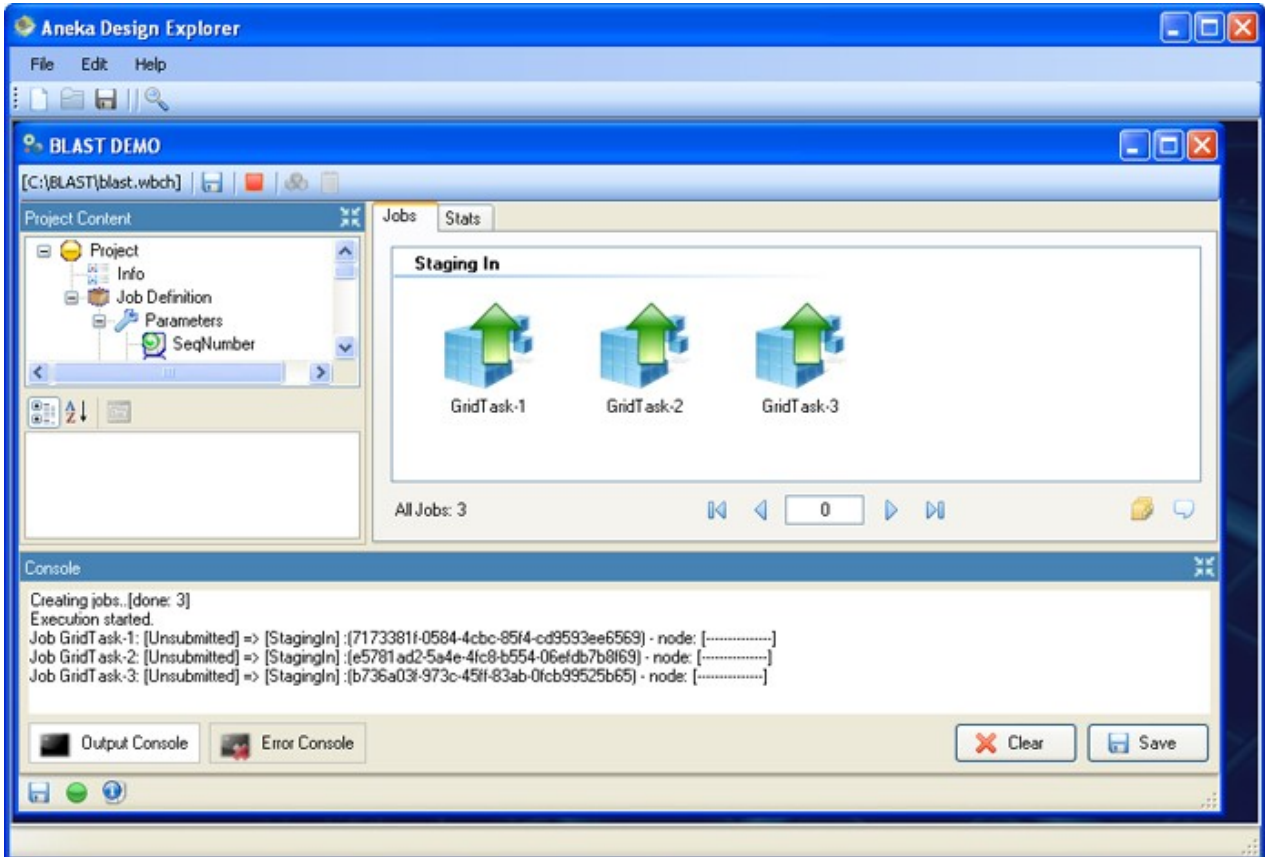


Figure 28. BLAST Demo Application Running.

Figure 29 shows the content of one of the result files obtained from the execution of the BLAST application.

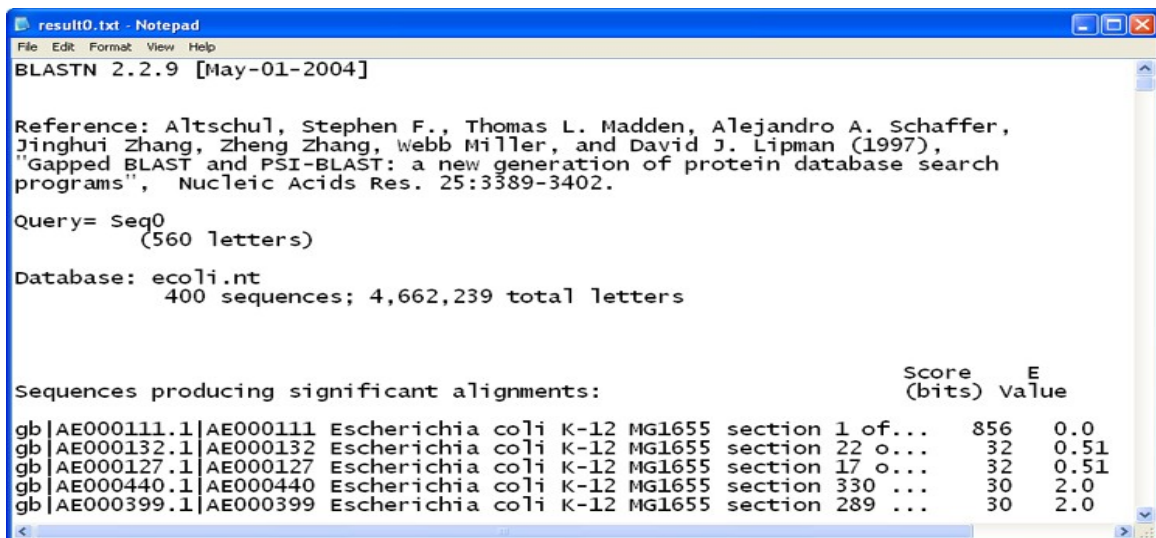


Figure 29. Content of the Result File.

The user can perform multiple runs by clicking again on the play button on the toolbar. As already introduced, for each of the execution a new subdirectory for the run is created. In this case there is no point in executing more runs of the application since the result of the search will be the same.

5.4 Extending the BLAST Example

As discussed in section 5.1.3 it is possible to increase the workload submitted to the Aneka Cloud it is possible to perform multiple searches on different databases. In this case the *Database* parameter can be characterized as enum parameter listing all the possible databases that will be searched. By changing the *Database* parameter from single to enum it is not possible to perform database formatting off line, it is then necessary to make that task part of each job. This means that the following changes have to be applied to the task template:

- configure the *formatdb.exe* executable as a shared file
- remove the *ecoli.nt.xxx* files as shared files (these will be generated by the task)
- configure the *Database* parameter as an enum parameter
- introduce the (*\$Database*) input file
- introduce first an EXE task executing *formatdb -i (\$Database) -p F -o T*
- leave the existing EXE task as it is.

The implementation of these changes is left to the reader.

6. Conclusions

In this tutorial we introduced the basic concepts concerning Parameter Sweeping Applications. We described the support provided by Aneka for implementing and managing this kind of applications and presented with major detail the Design Explorer.

The Design Explorer is integrated environment for prototyping and executing Parameter Sweeping applications on top of Aneka Clouds. It is a project based workspace and provides a user interface through which users can define the template task characterizing Parameter Sweeping Applications by using a simple step by step procedure. Task templates can be saved as plain XML documents within the project file or directly executed in the environment which allows to monitor the status of the application and collects some various statistics for the execution.

This tutorial has covered the following arguments:

- General notions about the Parameter Sweeping applications.
- Support provided by Aneka for running Parameter Sweeping applications (Parameter Sweeping Model).
- Overview of the Design Explorer features.

- How to create a simple Parameter Sweeping application with input and output files.
- How to manually create the XML file that represent the template task for the Parameter Sweeping Application without the support of the Design Explorer.
- How to compose the template task by using the command provided through the Design Explorer.
- How to monitor and control the execution of a Parameter Sweeping application.

All these features have been demonstrated by developing the *BLAST Demo* application from scratch.

This tutorial does not fully cover what can be done with the Parameter Sweeping Model that represents the set of APIs that Aneka exposes to developers for building Parameter Sweeping applications. In particular this tutorial did not explain how to compose by using the API a task template and generate and run tasks from it. For a more detailed information about this and other aspects the user can have a look at the corresponding APIs documentation (namespaces *Aneka.PSM.Core*, *Aneka.Tasks.BaseTasks*).