The (JAXB) Configurable Resource Manager for PTP

*Eclipse **Parallel Tools Platform** (http://eclipse.org/ptp)*

An introductory tutorial

June 22, 2011

Albert L. Rossi

National Center for Supercomputing Applications
arossi@ncsa.illinois.edu

What?

Allows you to launch and monitor applications on local or remote resources using a resource manager configured from an XML file via JAXB (*javax.xml.bind*) libraries.

Why?

1. Maximum adaptability: allow users to fit the resource manager to a class of systems, to a single host, or even to special application usage.
2. No Java coding is necessary. Users should be able to accommodate new systems without writing and loading additional Eclipse plugins.
3. Partition the client functionality so as to eliminate the need for special server-side proxies and to scale more successfully in the updating of job and resource information ("control" vs "monitor" components).

Configuring the JAXB Resource Manager: Outline

The following slides provide a brief introduction to the XML schema defining the JAXB Configurable Resource Manager, with a particular focus on the “control” component.

We first describe briefly the structure of the schema, along with how the resource manager “environment” is wired in the XML document.

We then offer a small step-by-step example in which we add new functionality to an existing resource manager; this will serve to illustrate a representative cross-section of the configurable features in the definition schema.

The Resource Manager Schema (XSD)

The Resource Manager XML Schema (*resource_manager_type.xsd*) is comprised of three principal sections:

<site-data>: used to provide fixed or default URLs for the control and monitor connections (optional).

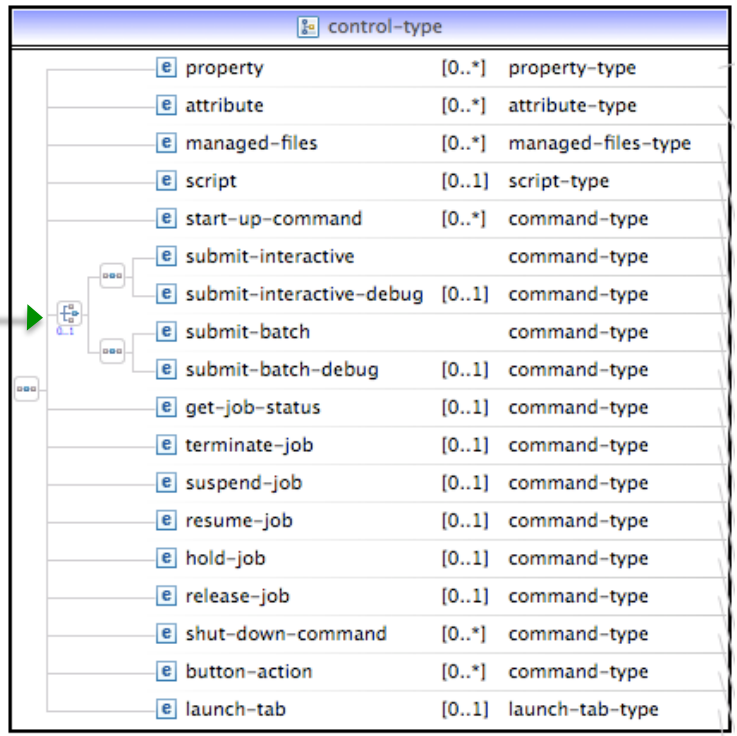
<control-data>: pertains to the "control" component which furnishes the job -control actions (submission, cancellation, status update, etc.); usually the part most often reconfigured or customized according to system, scheduler or the special needs of an application community.

<monitor-data>: settings necessary for configuring the ***LLview*** monitoring client (adjustments will be made less often to this part of the definition).

We will concentrate here on the second section of the schema.

The (Job) Control Component

A definition instance supports only one of these two pairs



properties = arbitrary variables internal to the client

attributes = "external" properties (such as those defined by the scheduler)

managed-files = local files, either pre-existent or written out from the definition itself, meant to be staged in conjunction with a job submission

script = for scheduler (batch) systems, the "batch script" (if any) to be staged in conjunction with a job submission

start-up-command(s) = arbitrary (remote) commands run when the resource manager is started

shut-down-command(s) = arbitrary (remote) commands run when the resource manager is stopped

submit-, terminate-, suspend-, resume-, hold-, release- = commands for controlling job submission

get-job-status = on-demand check of status of job

button-action = special command run via a Launch-Tab push-button

launch-tab = section describing the parts and disposition of the UI wizard used to configure and launch a job

A detailed guide to the XSD will be found in the Eclipse PTP developer documentation included as Help pages in the 5.0.1 release; it is currently available at http://wiki.eclipse.org/PTP/resource-managers .

The Resource Manager "Environment"

The *properties* and *attributes* defined in the XML, along with several preset properties provided internally, constitute the "environment" in which the JAXB Resource Manager runs.

When configuring the Resource Manager, this environment can be referenced in one of two ways.

- Some XML elements have attributes which take the name of the resource manager *attribute* or *property*.

  Ex. 1: parser adds entry to the *value* field (a List) of the *queues* property:

  ```
  <target ref="queues">
      ... <add field="value"> ...
  ```

  Ex. 2: combo sets the *value* field of *destination* to the selected item:

  ```
  <widget type="combo" style="SWT.BORDER" readOnly="true"
                                  saveValueTo="destination">
  ```

- A string value for the *property*'s or *attribute*'s fields can be obtained using the Eclipse variable resolution syntax. The namespace for the JAXB Resource Manager resolver is *ptp_rm*. The part preceding '#' indicates the name of the *property* or *attribute*, that following, the field:

  Ex.: tooltip on widget references that field of *destination* attribute:

  ```
  <tooltip>${ptp_rm:destination#tooltip}</tooltip>
  ```

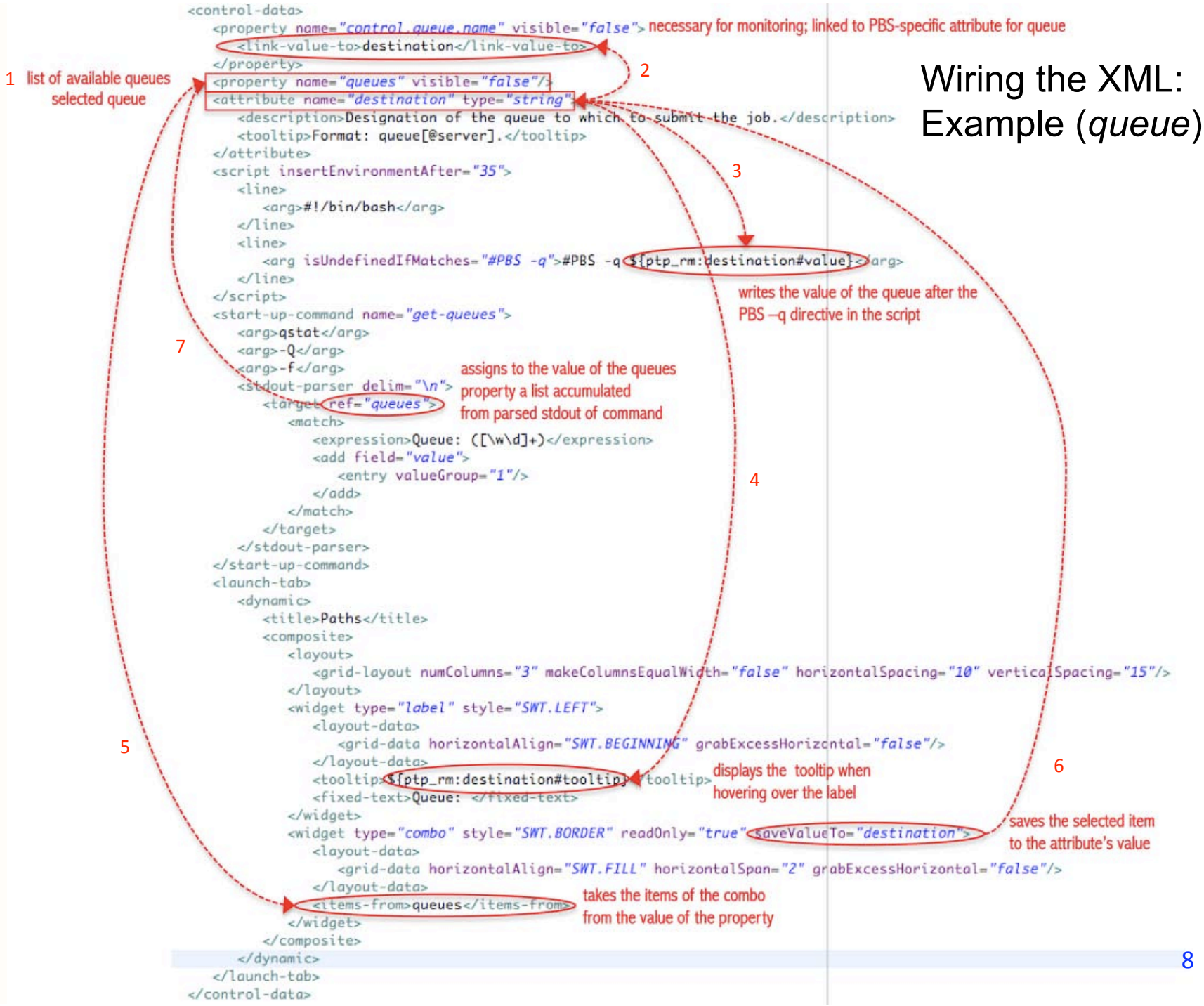
"Wiring" the Resource Manager Definition

A particularly powerful aspect of the Resource Manager's configurability derives from the ability, on the basis of this "environment", to make one part of the XML definition refer to another. We will call this "wiring" the definition.

On the next two slides, we present and explain an example of this procedure, based on the setting and reading of variables related to the choice of the scheduler queue.

On the slide following this discussion, we will then note two special procedures necessary for capturing certain values present only after you select "Run" and launch the job.

Wiring the XML: Example (*queue*)



"Wiring" the Resource Manager Definition

1. There are two main "variables" for handling the queue name: *destination*, which points to the actual queue chosen, and *queues*, which provides a list of available queues.
2. The *control.queue.name* is an internal property used under the covers to convey information to the monitor. The <link-value-to> element means that this property's value is set to that of *destination*, or in the case that *destination*'s value is undefined, to any default value given to *control.queue.name* (here none).
3. The script element has a line whose only argument references the value of *destination*; if this is empty, the argument will resolve to "#PBS –q", and since it is indicated that this should be considered equivalent to undefined, the argument will be eliminated.
4. The *tooltip* string is given to the label widget pointing to the combo list where the queue can be selected.
5. The combo list widget itself takes its preset values (notice that it is "readOnly", so the user cannot type in a value here but is constrained to the provided choices) from the list of available *queues*.
6. The selection made via the combo widget will become the value of the *destination* attribute.
7. The start-up command runs "qstat –Q –f", then uses a regular expression to parse the standard output and to accumulate the matching values as entries in a List which it sets as the value of the *queues* property.

"Wiring" the Resource Manager Definition

SPECIAL NOTES: ***visible, @jobId, <managed-file> target paths***

The ***visible*** attribute on a *property* or *attribute* is a way of indicating to the Resource Manager that the user will not be directly changing its value via the Launch Tab interface. Certain widgets (such as the attribute table or tree) check this to see if the *property* or *attribute* should be included automatically in its list.

@jobId: This is a special property name designating the runtime id for a job instance. In the lifecycle of the run/launch call, this value begins as an internally generated unique id which then is swapped for the id returned by the scheduler.

The @jobId, along with the target paths for <managed-file> elements, are not known at configuration time (i.e., before the user hits "Run"). While the former is made visible to the parsers and the returned status object of the submit command, neither is available for reference in other managed files or in the <script> element, because these elements are generated prior to the actual submission.

If the <script> needs to refer to the *@jobId*, it must do so via the variable made available by the particular scheduler it is written for. An example of *how to reference the target paths of other <managed-file>s inside the script* is included in the illustration which follows.

Customizing the Resource Manager Definition: An Illustration

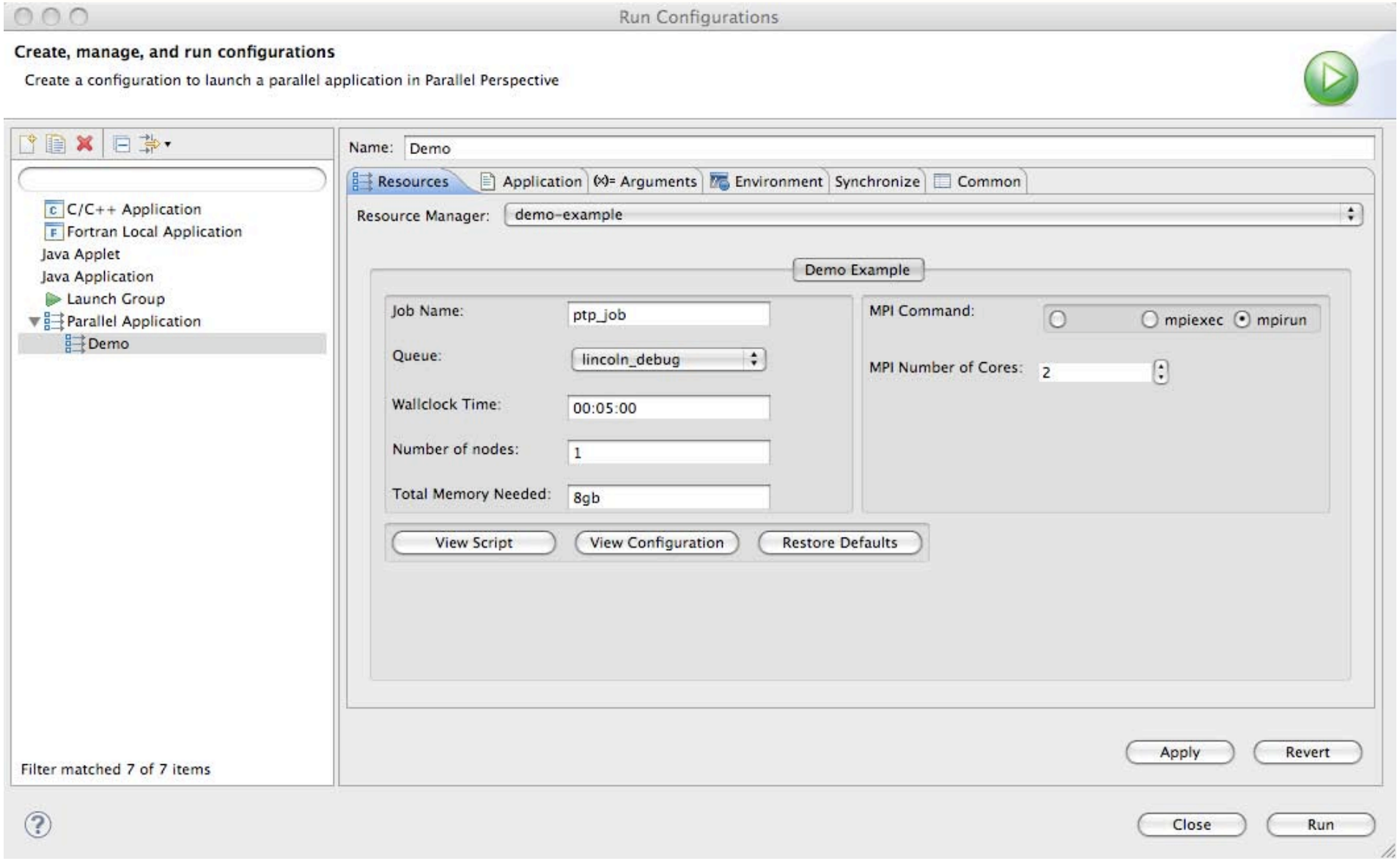
In the following illustration, we will take a pre-existing Resource Manager definition and modify it by adding functionality to support a particular application scenario.

Let us suppose we wish to tailor the definition for use with a simulation code which requires an input file; we would like the user to be able to choose this file before launching the job.

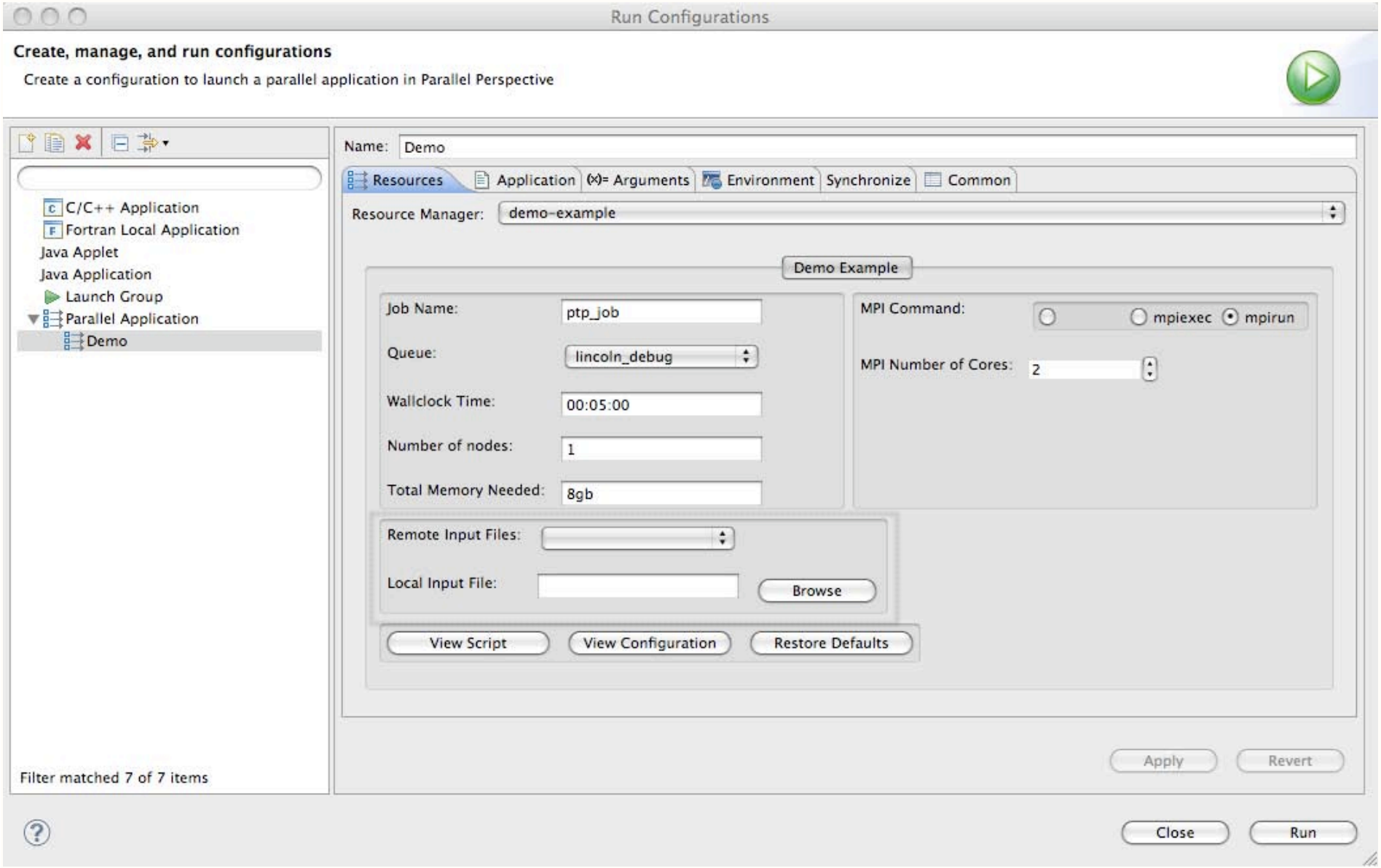
Let us further say that the file could be chosen either from a predetermined location on the remote host, or could be selected from a file edited locally.

In the former case, we would simply need to select a (remote) path, whereas in the latter, we would actually need to stage the file over before executing the call to submit the job.

Before


Run Configurations
Create, manage, and run configurations
Create a configuration to launch a parallel application in Parallel Perspective
C/C++ Application
Fortran Local Application
Java Applet
Java Application
Launch Group
Parallel Application
Demo
Filter matched 7 of 7 items
Name: Demo
Resources
Application
(x)= Arguments
Environment
Synchronize
Common
Resource Manager: demo-example
Demo Example
Job Name: ptp_job
Queue: lincoln_debug
Wallclock Time: 00:05:00
Number of nodes: 1
Total Memory Needed: 8gb
MPI Command: mpiexec mpirun
MPI Number of Cores: 2
View Script
View Configuration
Restore Defaults
Apply
Revert
Close
Run

After


Run Configurations
Create, manage, and run configurations
Create a configuration to launch a parallel application in Parallel Perspective
C/C++ Application
Fortran Local Application
Java Applet
Java Application
Launch Group
Parallel Application
Demo
Filter matched 7 of 7 items
Name: Demo
Resources
Application
(x)= Arguments
Environment
Synchronize
Common
Resource Manager: demo-example
Demo Example
Job Name: ptp_job
Queue: lincoln_debug
Wallclock Time: 00:05:00
Number of nodes: 1
Total Memory Needed: 8gb
MPI Command: mpiexec mpirun
MPI Number of Cores: 2
Remote Input Files:
Local Input File:
Browse
View Script
View Configuration
Restore Defaults
Apply
Revert
Close
Run

Customizing the Resource Manager: Steps

What we will add to the existing definition:

1. Three **properties** for handling the paths.
2. Two widgets:
   a. A **combo** list populated from the contents of a remote directory;
   b. A **browse text + button** for selecting a local file.
3. A **managed file** for staging in case of 2b being selected.
4. A **start-up command** and **parser** which will populate the items of 2a.
5. An **environment variable** for conveying the path to the batch script.
6. The additional **reference to the path** on the execution line of the batch script.

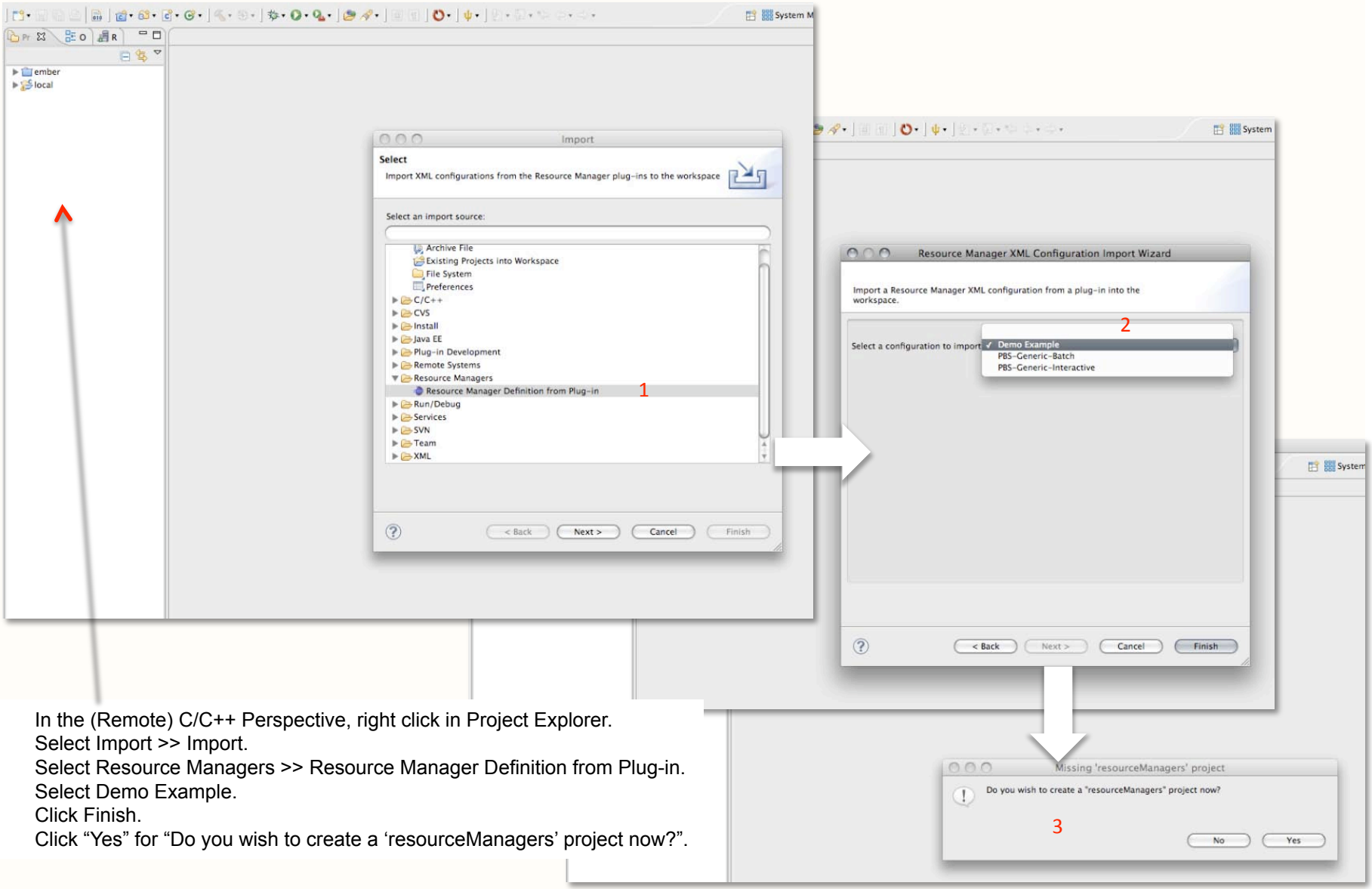
Customizing the Resource Manager:
Getting the base .xml file

First, we will need to import the provided definition file into our workspace for editing.

Note: "Demo Example" is not provided in the Indigo release; it can be downloaded from:

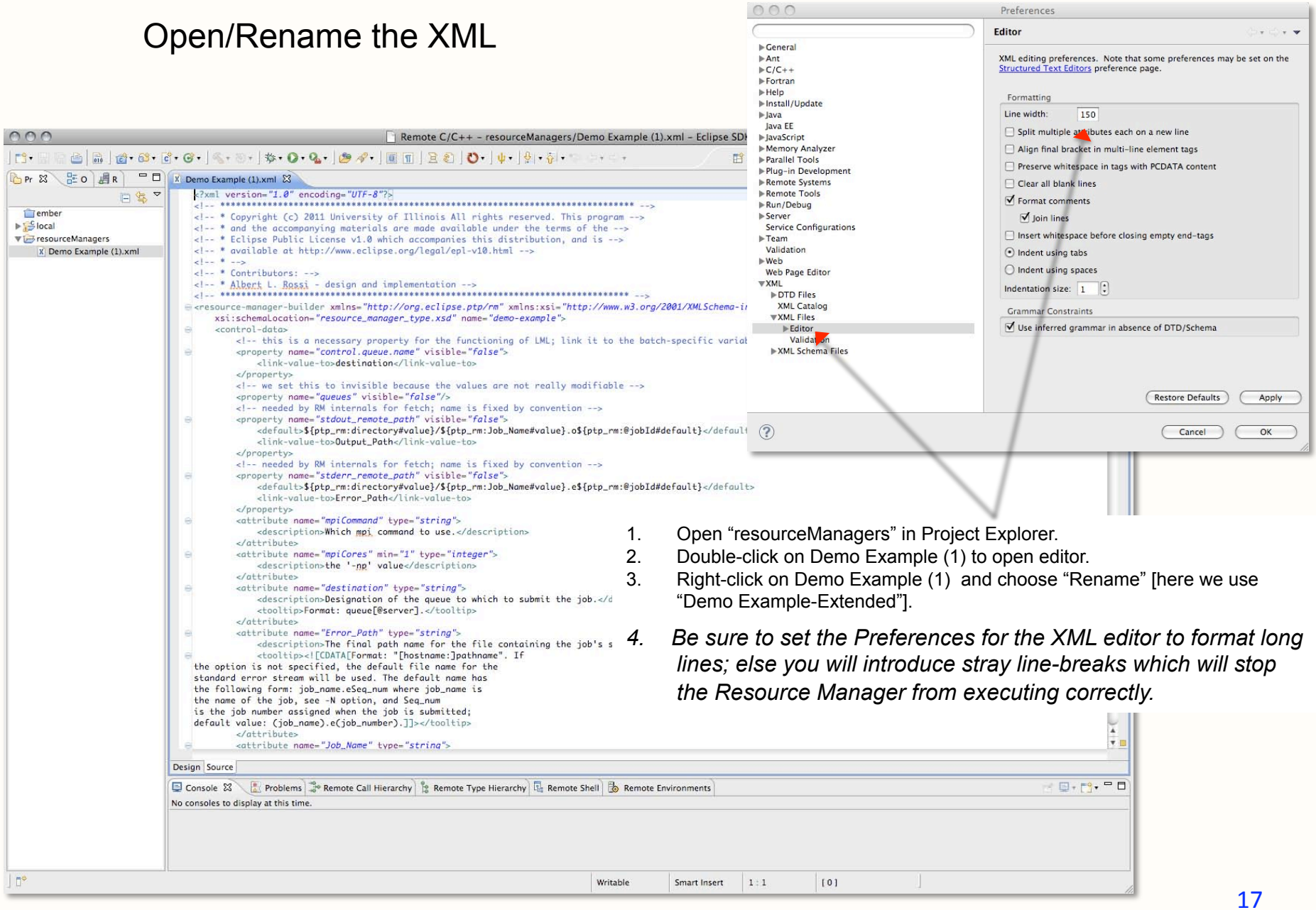
http://wiki.eclipse.org/PTP/resource-managers#Configuring.2FCustomizing_the_Resource_Manager

Import the XML into your Project Workspace



1. In the (Remote) C/C++ Perspective, right click in Project Explorer.
2. Select Import >> Import.
3. Select Resource Managers >> Resource Manager Definition from Plug-in.
4. Select Demo Example.
5. Click Finish.
6. Click “Yes” for “Do you wish to create a ‘resourceManagers’ project now?”.

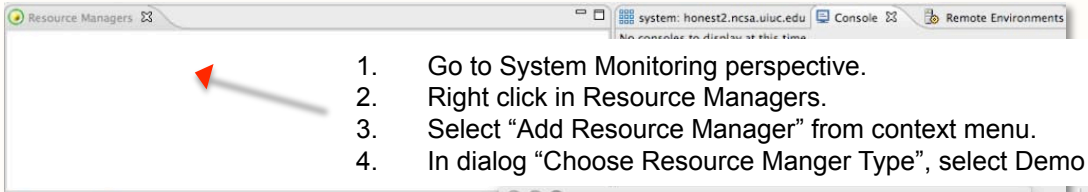
Open/Rename the XML



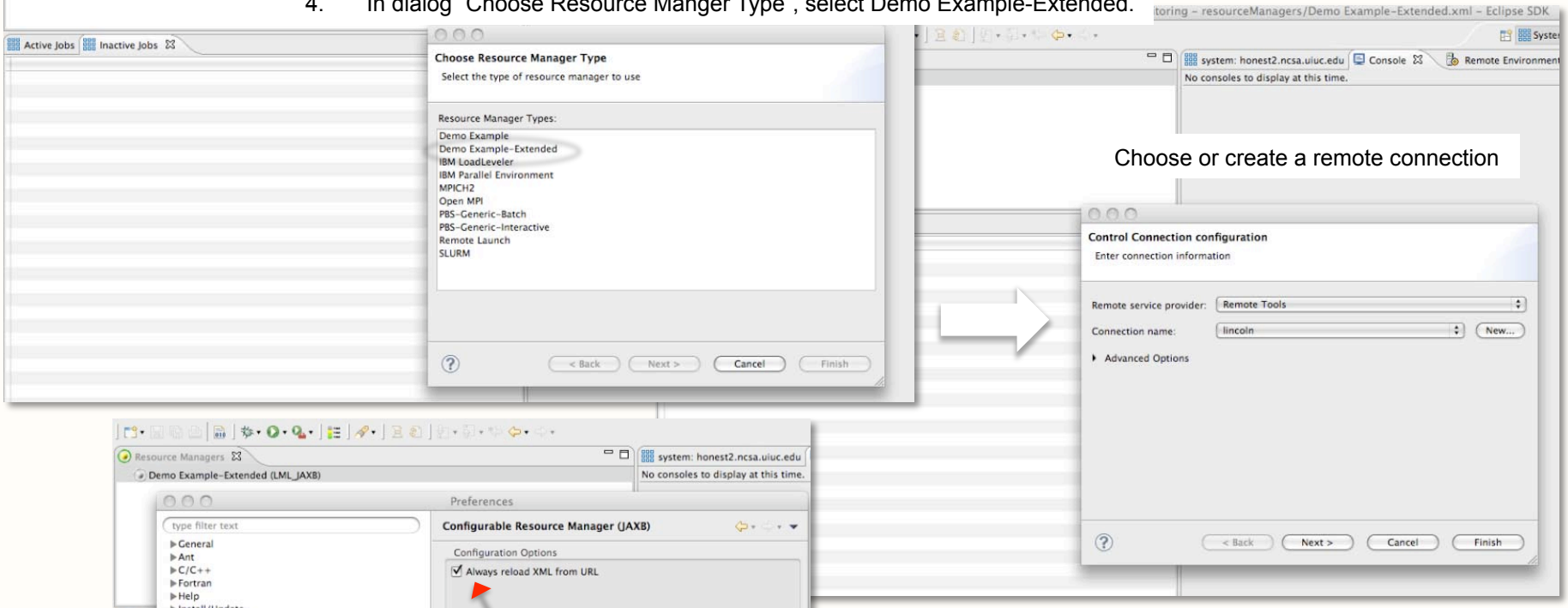
1. Open “resourceManagers” in Project Explorer.
2. Double-click on Demo Example (1) to open editor.
3. Right-click on Demo Example (1) and choose “Rename” [here we use “Demo Example-Extended”].

4. *Be sure to set the Preferences for the XML editor to format long lines; else you will introduce stray line-breaks which will stop the Resource Manager from executing correctly.*

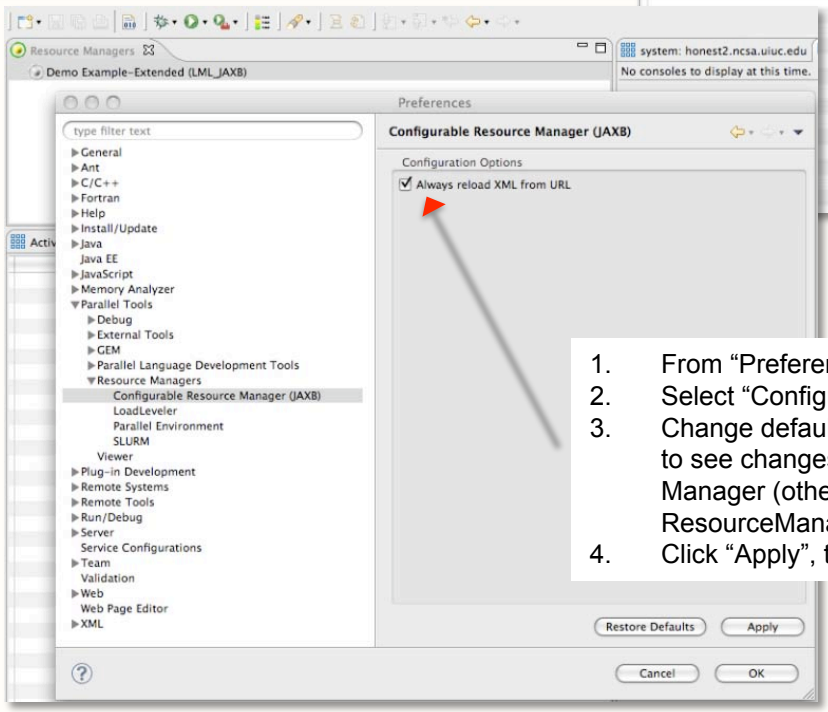
Add the Resource Manager



1. Go to System Monitoring perspective.
2. Right click in Resource Managers.
3. Select “Add Resource Manager” from context menu.
4. In dialog “Choose Resource Manger Type”, select Demo Example-Extended.



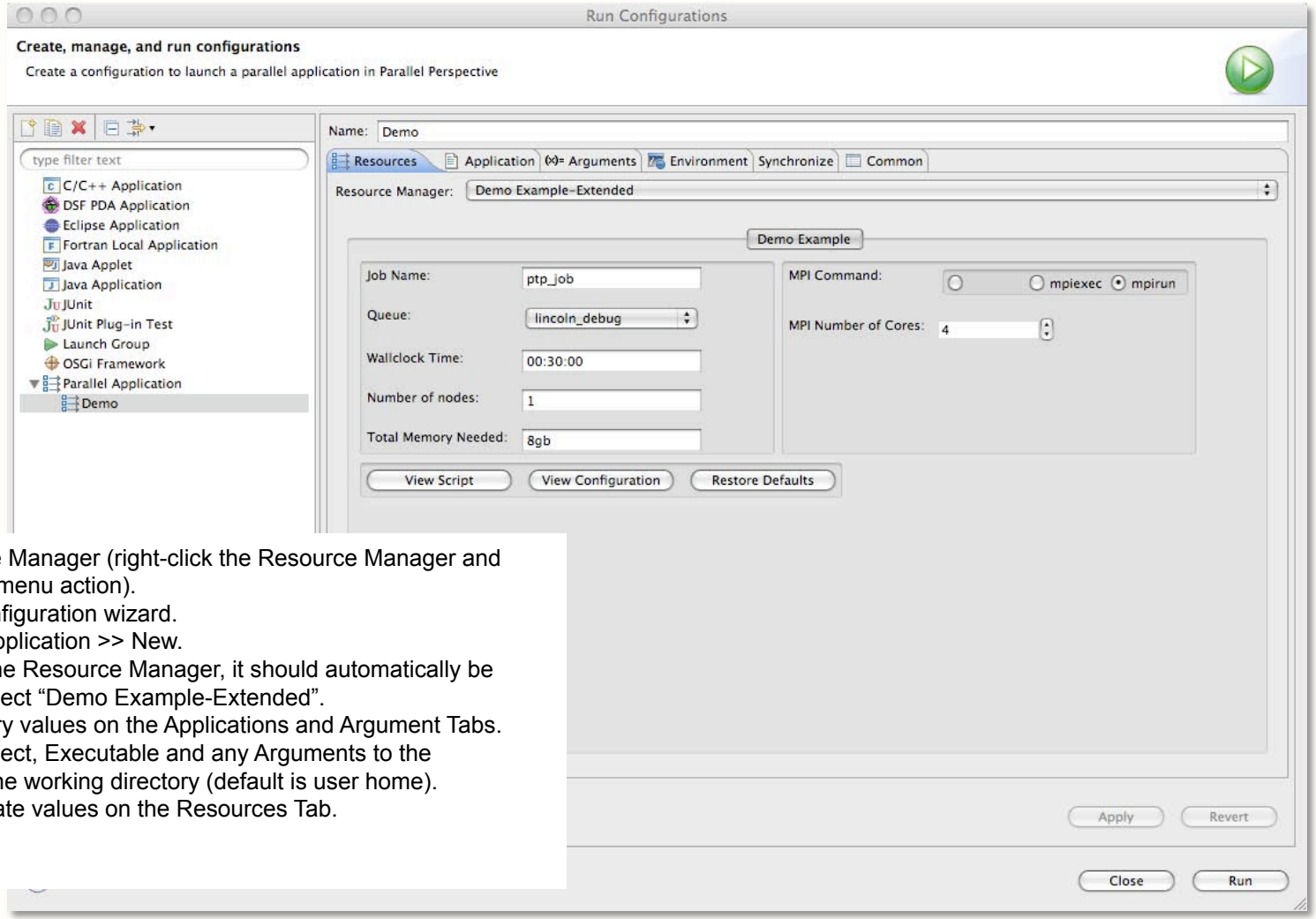
Choose or create a remote connection



1. From “Preferences” Menu, select Parallel Tools >> Resource Managers.
2. Select “Configurable Resource Manager”.
3. Change default setting (unchecked) for “Always reload”; this allows you to see changes made to the XML each time you restart the Resource Manager (otherwise, the file is cached and reused for the life of the ResourceManager).
4. Click “Apply”, then “OK”.

Launch Tab Set-Up

Proceed with the usual Launch Tab configuration.



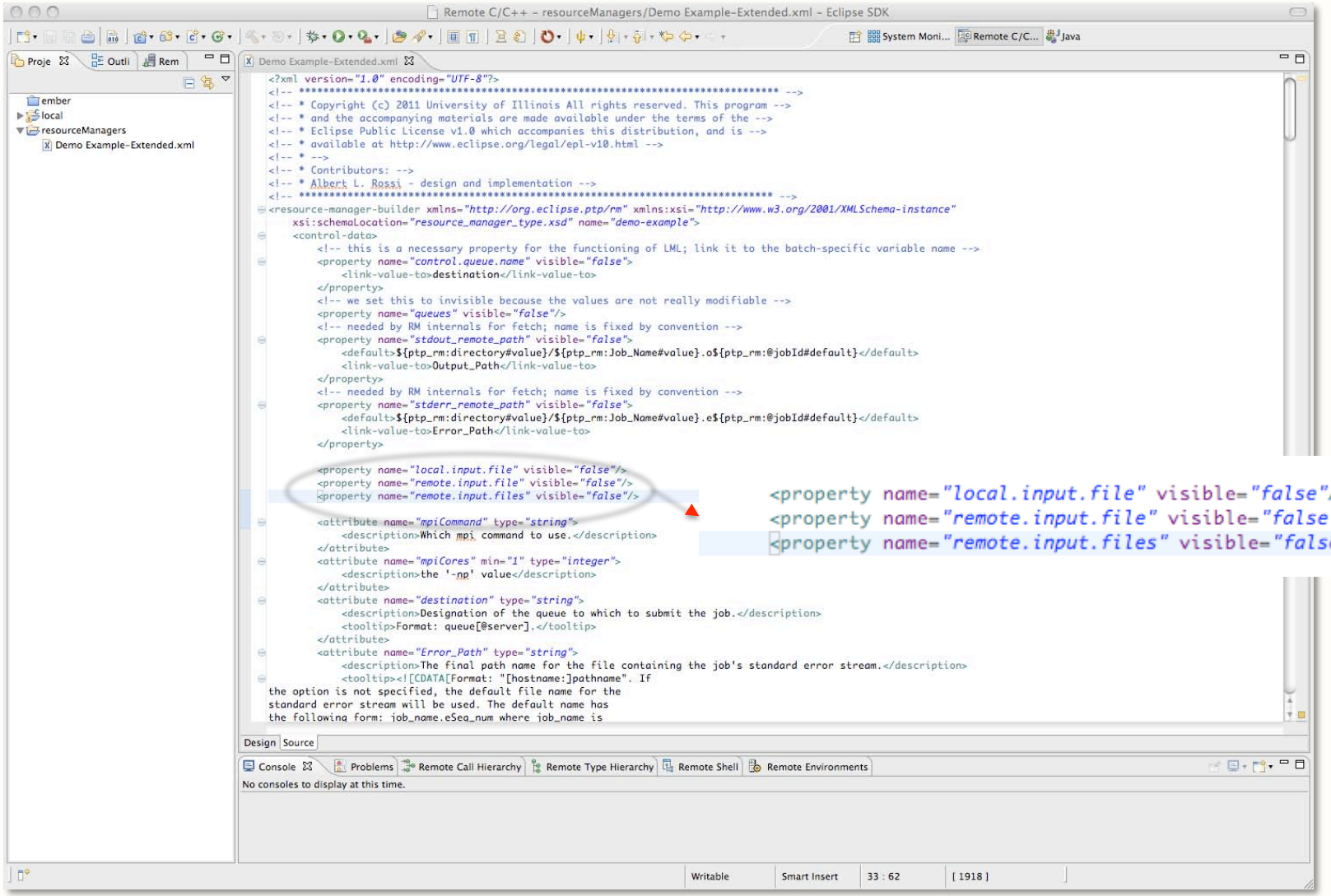
1. Start the Resource Manager (right-click the Resource Manager and select the context menu action).
2. Open the Run Configuration wizard.
3. Choose Parallel Application >> New.
4. If you have only one Resource Manager, it should automatically be selected; if not, select “Demo Example-Extended”.
5. Fill in the necessary values on the Applications and Argument Tabs. These include Project, Executable and any Arguments to the Executable, plus the working directory (default is user home).
6. Fill in the appropriate values on the Resources Tab.
7. Click Apply.

You now have the basic tab which we will proceed to modify.

This Resource Manager provides basic PBS batch settings, and on start-up looks for the available queues. The actions enabled are for job submission, getting job status and job cancellation.

1. Properties

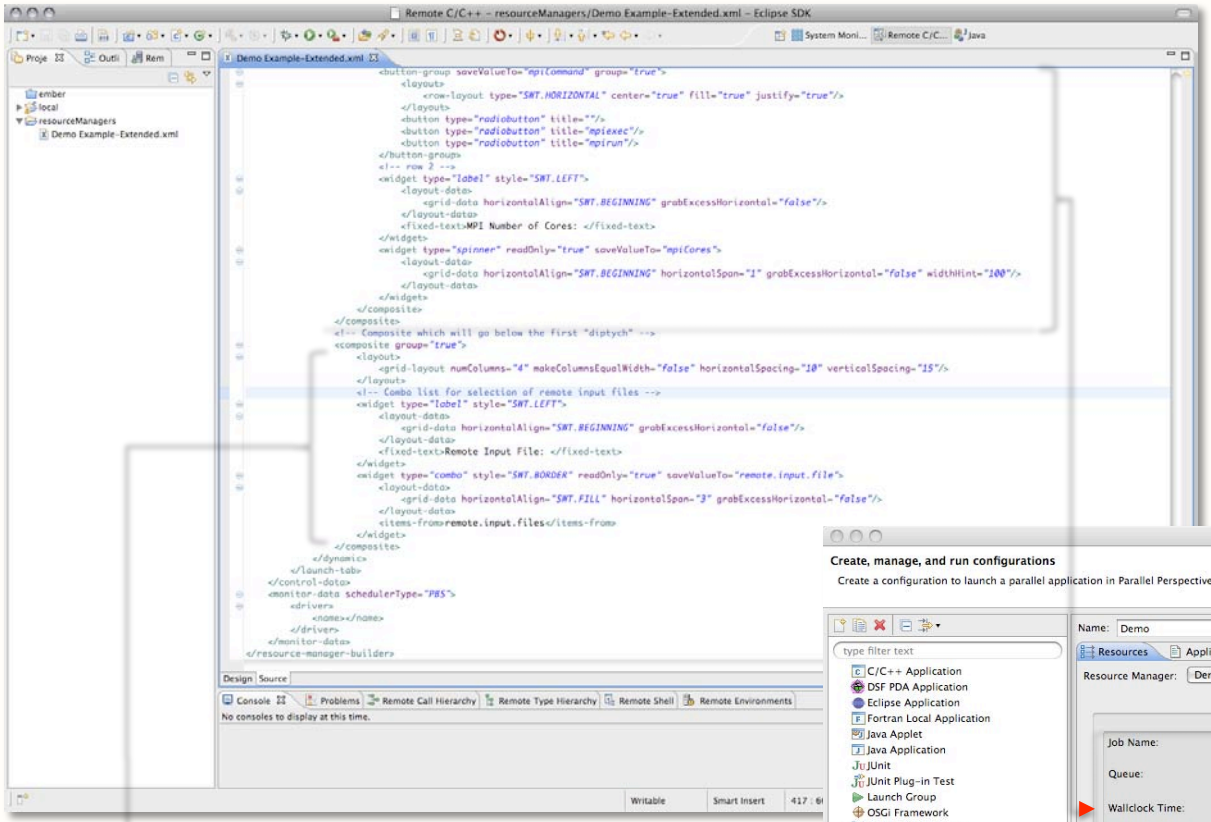
Add three properties, two to reference the path choice, and a third to hold a list of remote paths.

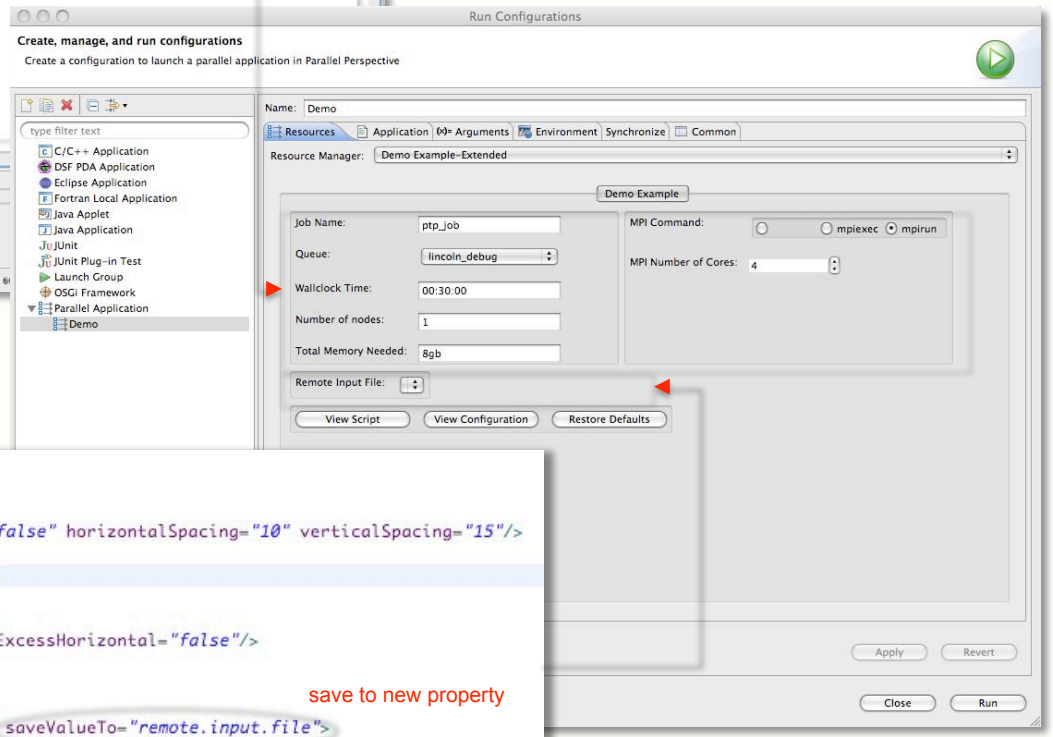


2a. Combo Box

Add group with 4 columns below main composite of two panels.

Add label + 3-column combo.





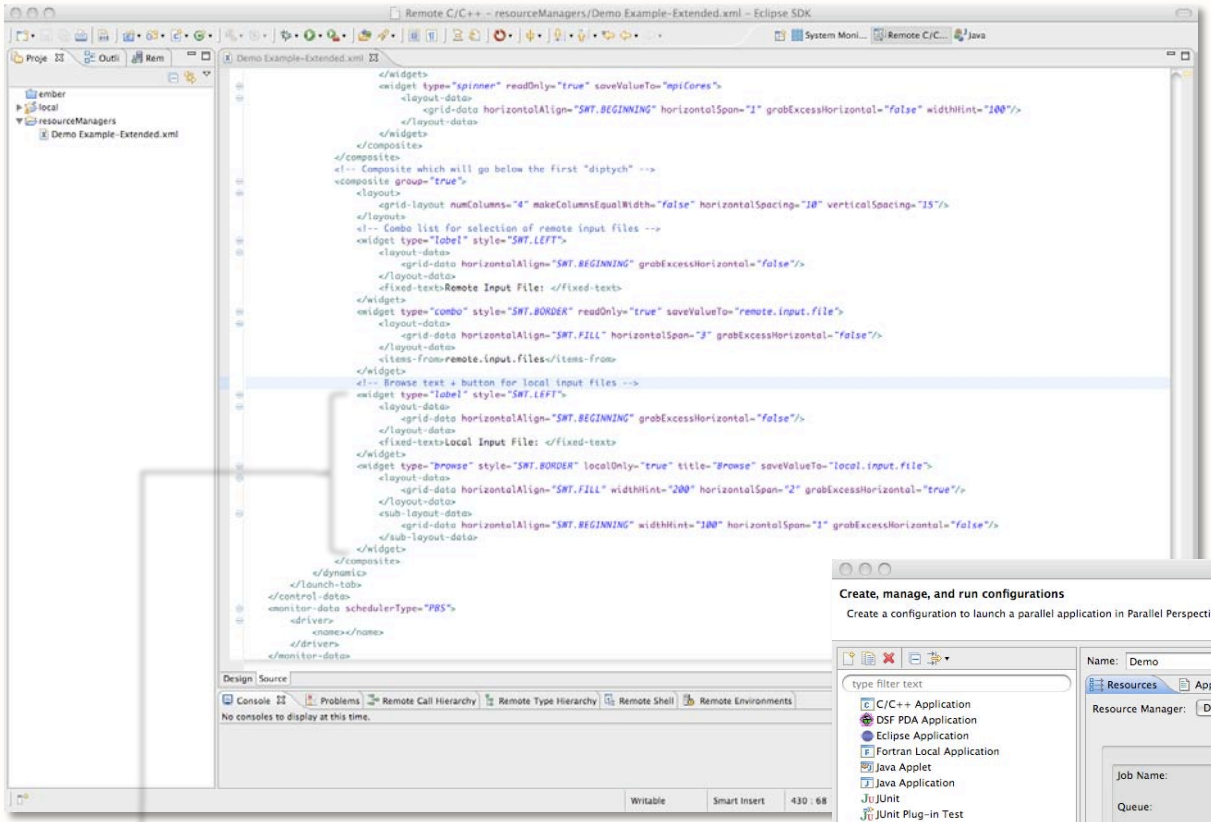

```
<!-- Composite which will go below the first "diptych" -->
<composite group="true">
    <layout>
        <grid-layout numColumns="4" makeColumnsEqualWidth="false" horizontalSpacing="10" verticalSpacing="15"/>
    </layout>
    <!-- Combo list for selection of remote input files -->
    <widget type="label" style="SWT.LEFT">
        <layout-data>
            <grid-data horizontalAlign="SWT.BEGINNING" grabExcessHorizontal="false"/>
        </layout-data>
        <fixed-text>Remote Input File: </fixed-text>
    </widget>
    <widget type="combo" style="SWT.BORDER" readOnly="true" saveValueTo="remote.input.file">
        <layout-data>
            <grid-data horizontalAlign="SWT.FILL" horizontalSpan="3" grabExcessHorizontal="false"/>
        </layout-data>
        <items-from>remote.input.files</items-from>
    </widget>
</composite>
```

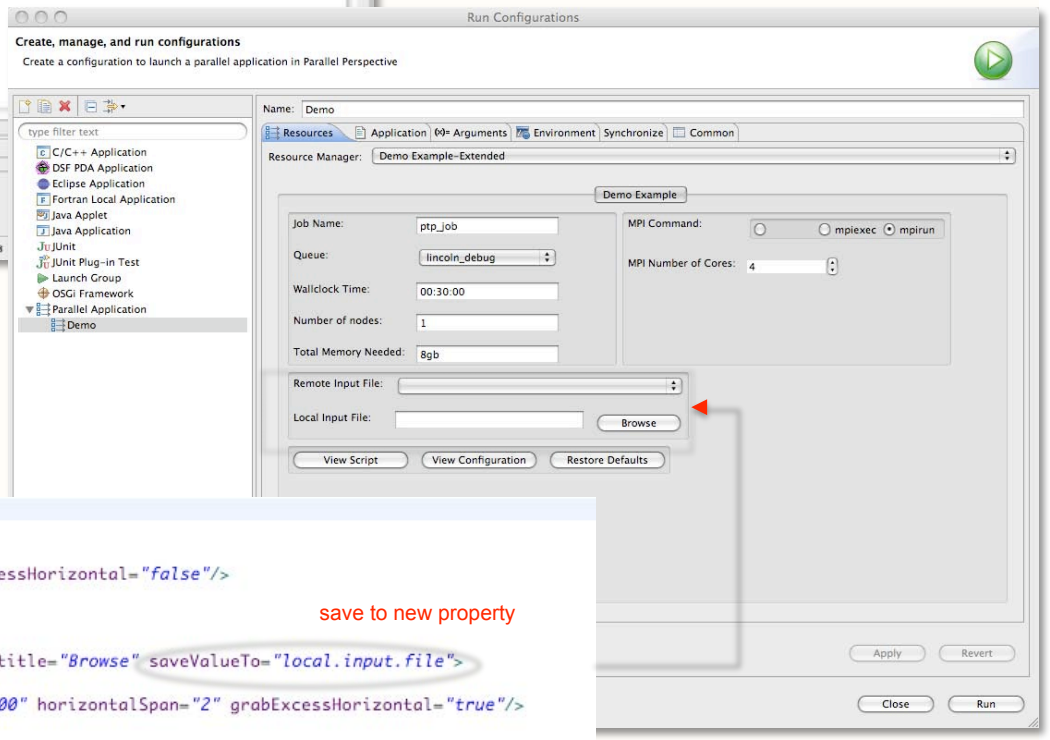
save to new property

get items from new property

2b. Browse Text + Button

Add label + this paired widget (text gets 2 columns) after the combo box inside the new composite/group.





Note that the browse button is set to *localOnly*="true" (it will access only the local file system).

We *do not* force *readOnly*="true" here so that one can clear and edit the text box if so desired.

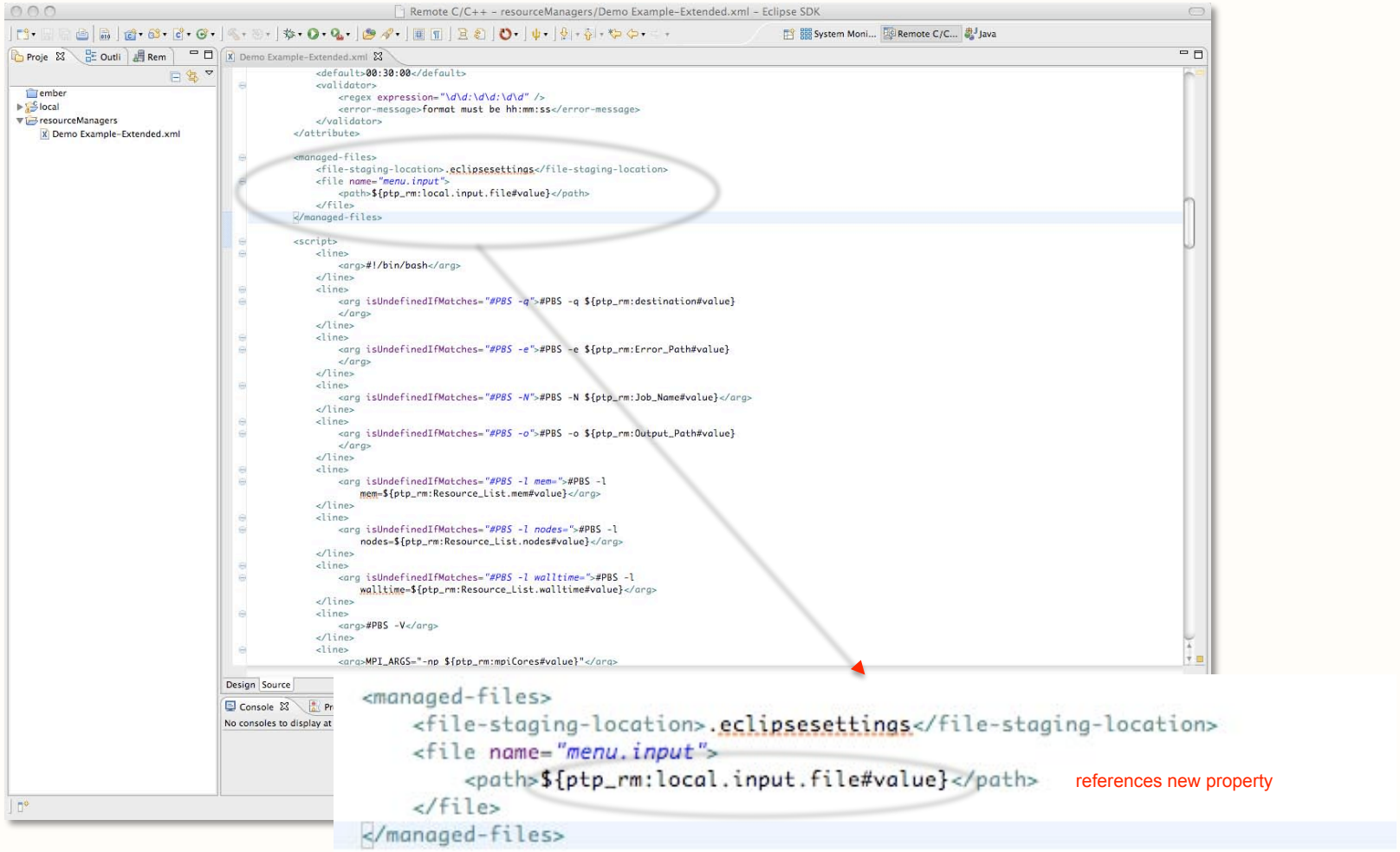
```
<!-- Browse text + button for local input files -->
<widget type="label" style="SWT.LEFT">
    <layout-data>
        <grid-data horizontalAlign="SWT.BEGINNING" grabExcessHorizontal="false"/>
    </layout-data>
    <fixed-text>Local Input File: </fixed-text>
</widget>
<widget type="browse" style="SWT.BORDER" localOnly="true" title="Browse" saveValueTo="local.input.file">
    <layout-data>
        <grid-data horizontalAlign="SWT.FILL" widthHint="200" horizontalSpan="2" grabExcessHorizontal="true"/>
    </layout-data>
    <sub-layout-data>
        <grid-data horizontalAlign="SWT.BEGINNING" widthHint="100" horizontalSpan="1" grabExcessHorizontal="false"/>
    </sub-layout-data>
</widget>
```

save to new property

layout data for text

layout data for button

3. Managed File

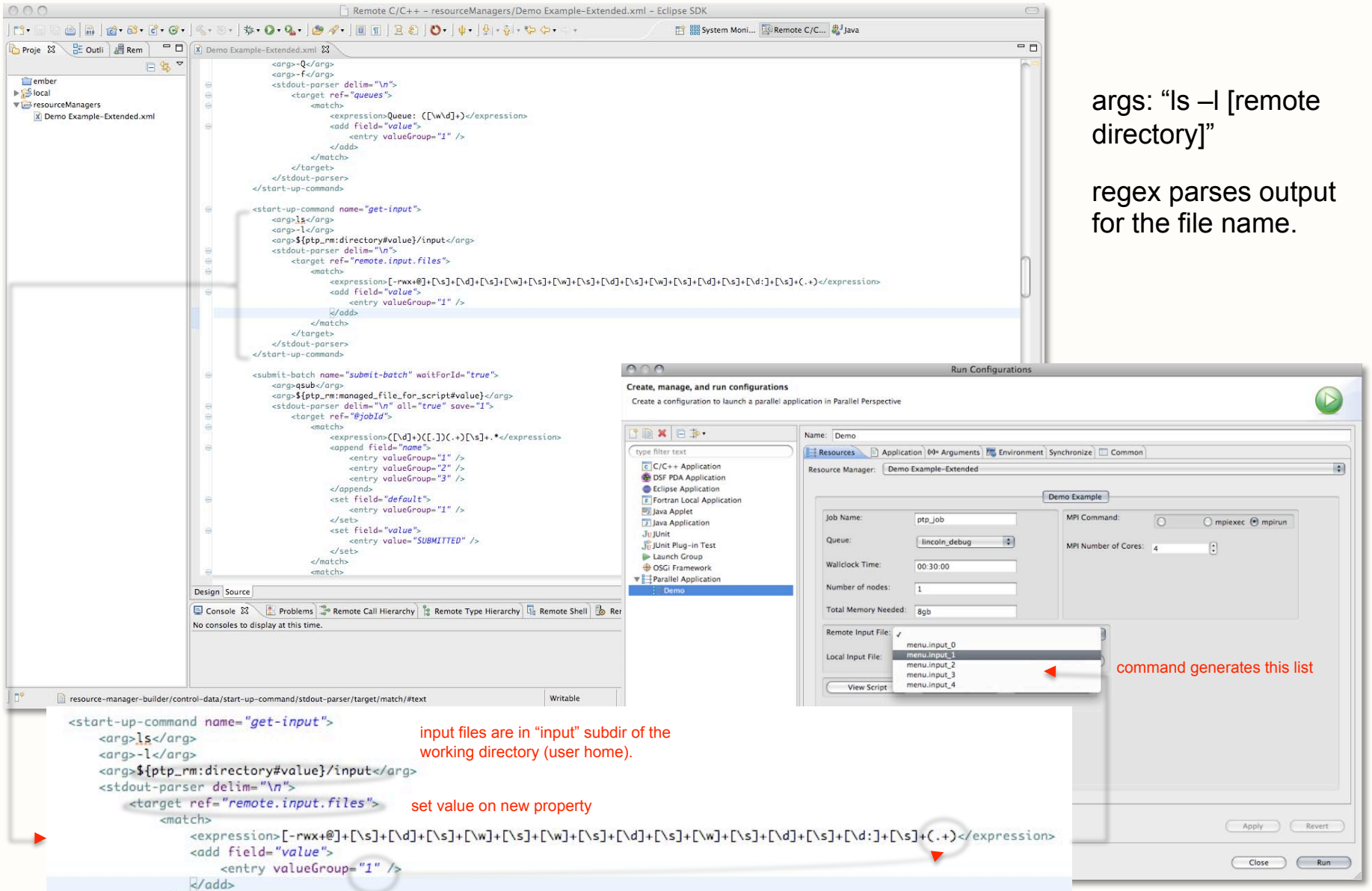


- Local path references the new property set from the browse text-widget field.
- Stages the file to ".eclipsesettings" in user home.
- Remote path will be *.eclipsesettings/local.input.file.name*. This is accessed in the environment as: *${ptp_rm:menu.input#value}*.

4. Start-up command

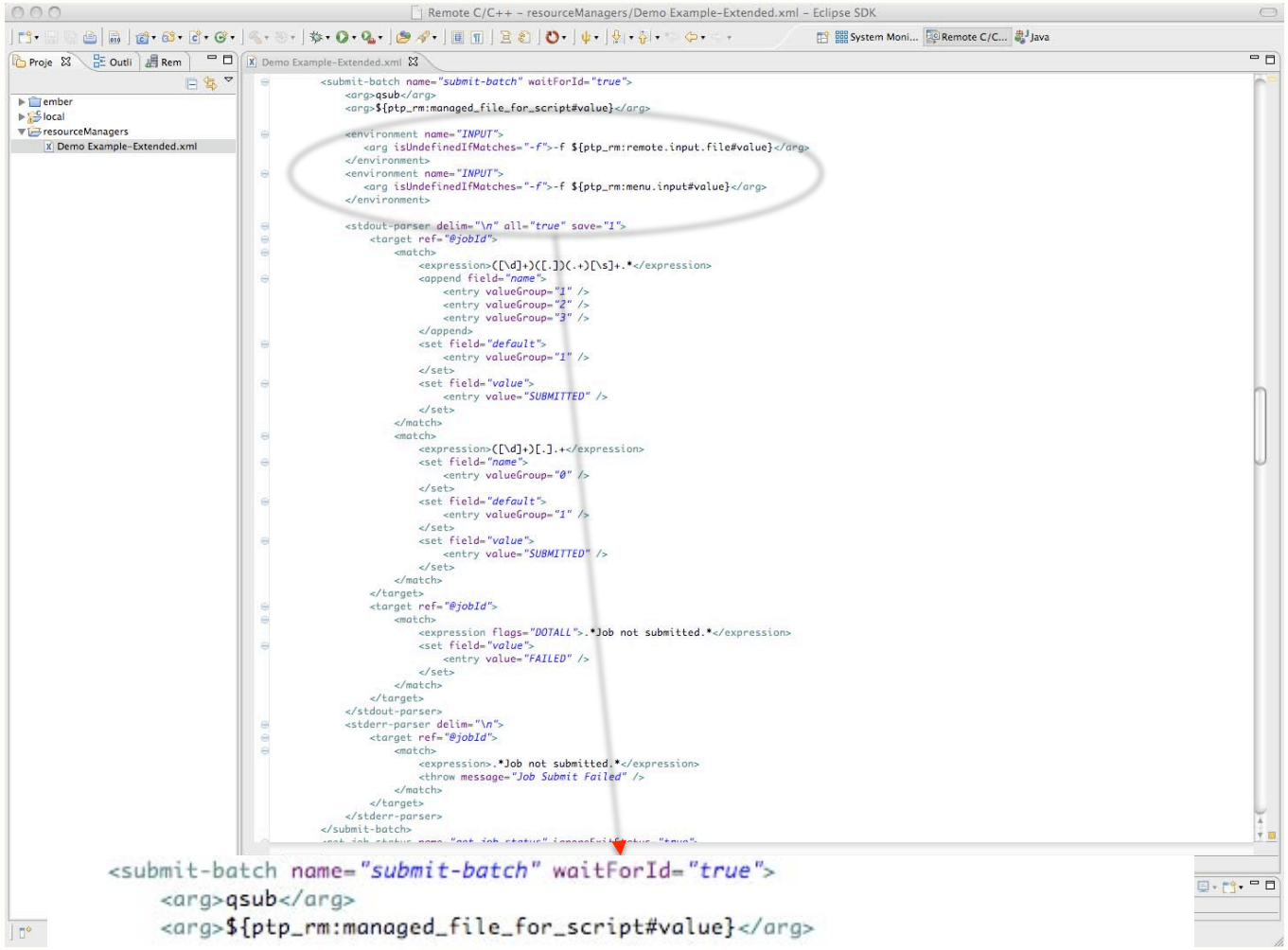
args: “ls –l [remote directory]”

regex parses output for the file name.




```
<start-up-command name="get-input">
    <arg>ls</arg>
    <arg>-l</arg>
    <arg>${ptp_rm:directory#value}/input</arg>
    <stdout-parser delim="\n">
        <target ref="remote.input.files">
            <match>
                <expression>[-rwx+@]+[\s]+[\d]+[\s]+[\w]+[\s]+[\w]+[\s]+[\d]+[\s]+[\w]+[\s]+[\d]+[\s]+[\d:]+[\s]+(.+)</expression>
                <add field="value">
                    <entry valueGroup="1" />
                </add>
            </match>
        </target>
    </stdout-parser>
</start-up-command>
```

5. Adjust Environment of Submit Command




```
<submit-batch name="submit-batch" waitForId="true">
    <arg>qsub</arg>
    <arg>${ptp_rm:managed_file_for_script#value}</arg>

    <environment name="INPUT">
       <arg isUndefinedIfMatches="-f">-f ${ptp_rm:remote.input.file#value}</arg>
    </environment>
    <environment name="INPUT">
       <arg isUndefinedIfMatches="-f">-f ${ptp_rm:menu.input#value}</arg>
    </environment>
```

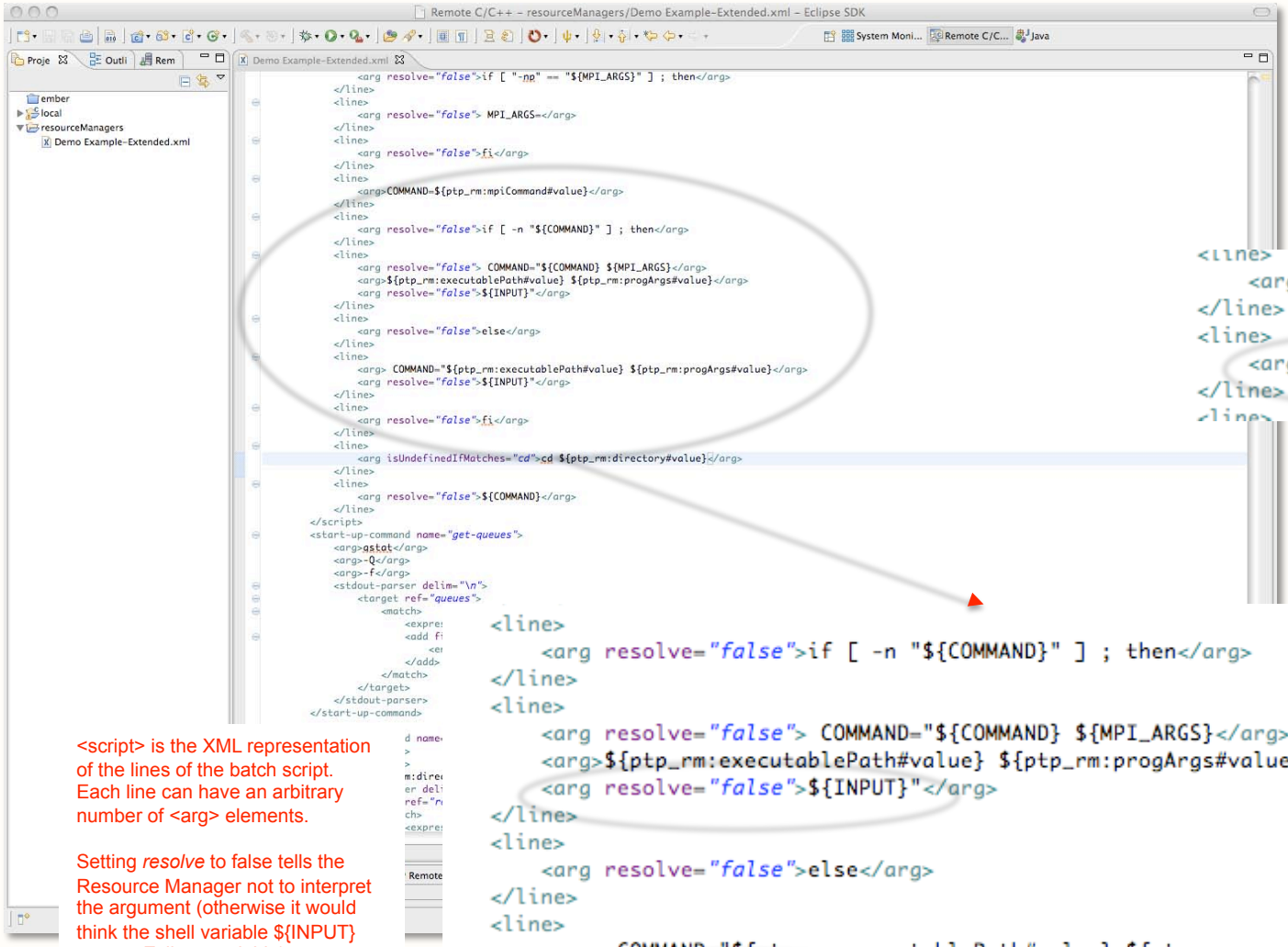
references new property

references managed file property (remote path); should overwrite first INPUT definition *only* if defined
(note: a bug in the managed file code, *fixed in release 5.0.1*, was setting this path to the staging directory when the actual file was undefined)

Because the managed files are configured after “Run” is selected, but just before the actual remote submission of the job, the remote target path can be captured and placed in the job’s environment, making it available to the batch script when it becomes active.

Environment definitions follow the command args; the value of an environment variable can be expressed via its “value” attribute, or as embedded <arg>s, as it is here.

6. Adjust Script Execution Line



Add ${INPUT} argument for the input file.

```
<line>
    <arg isUndefinedIfMatches="
</line>
<line>
    <arg>#PBS -V</arg>
</line>
```

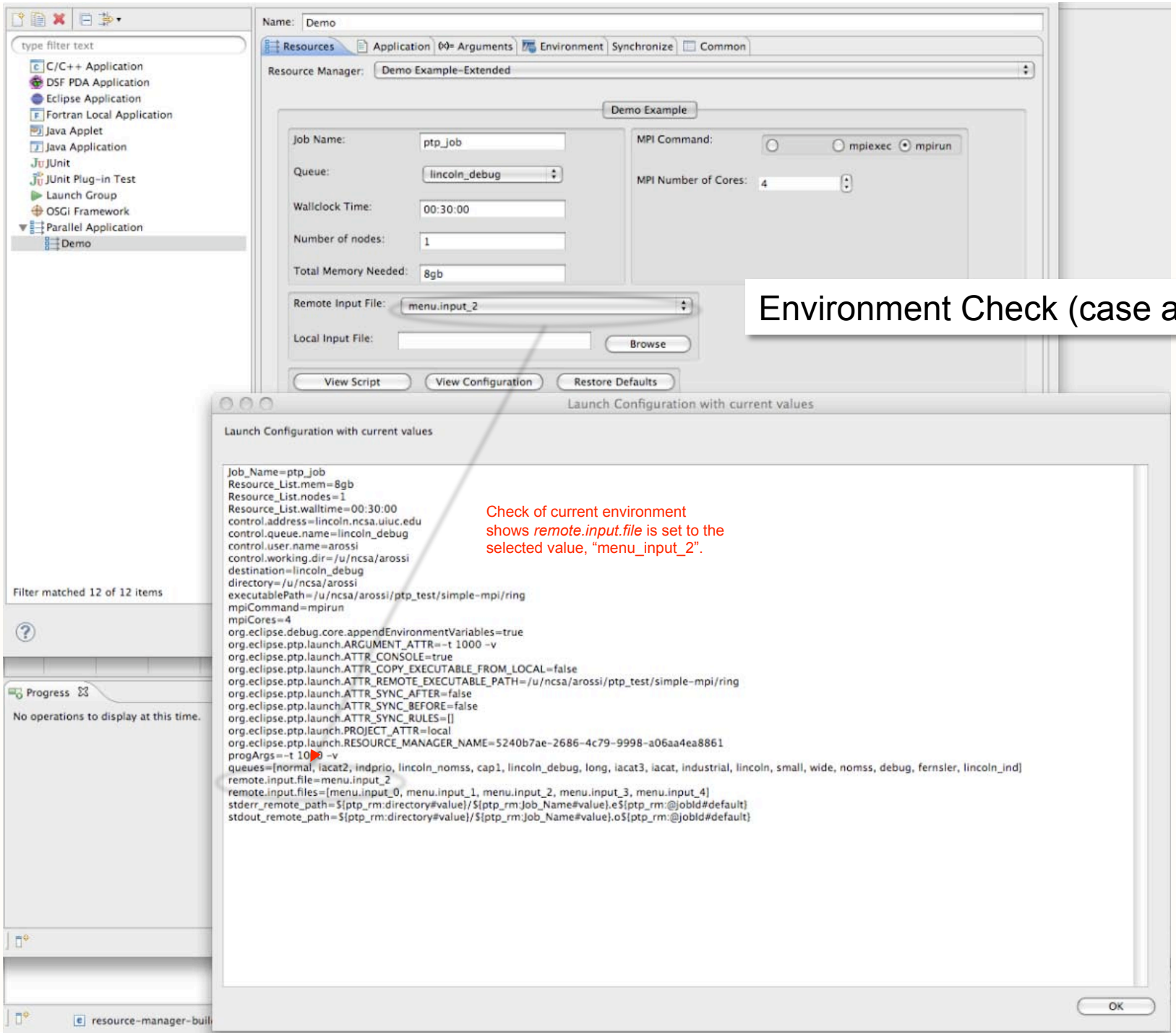
Be sure that your script indicates that the environment be passed on to the child processes.

<script> is the XML representation of the lines of the batch script. Each line can have an arbitrary number of <arg> elements.

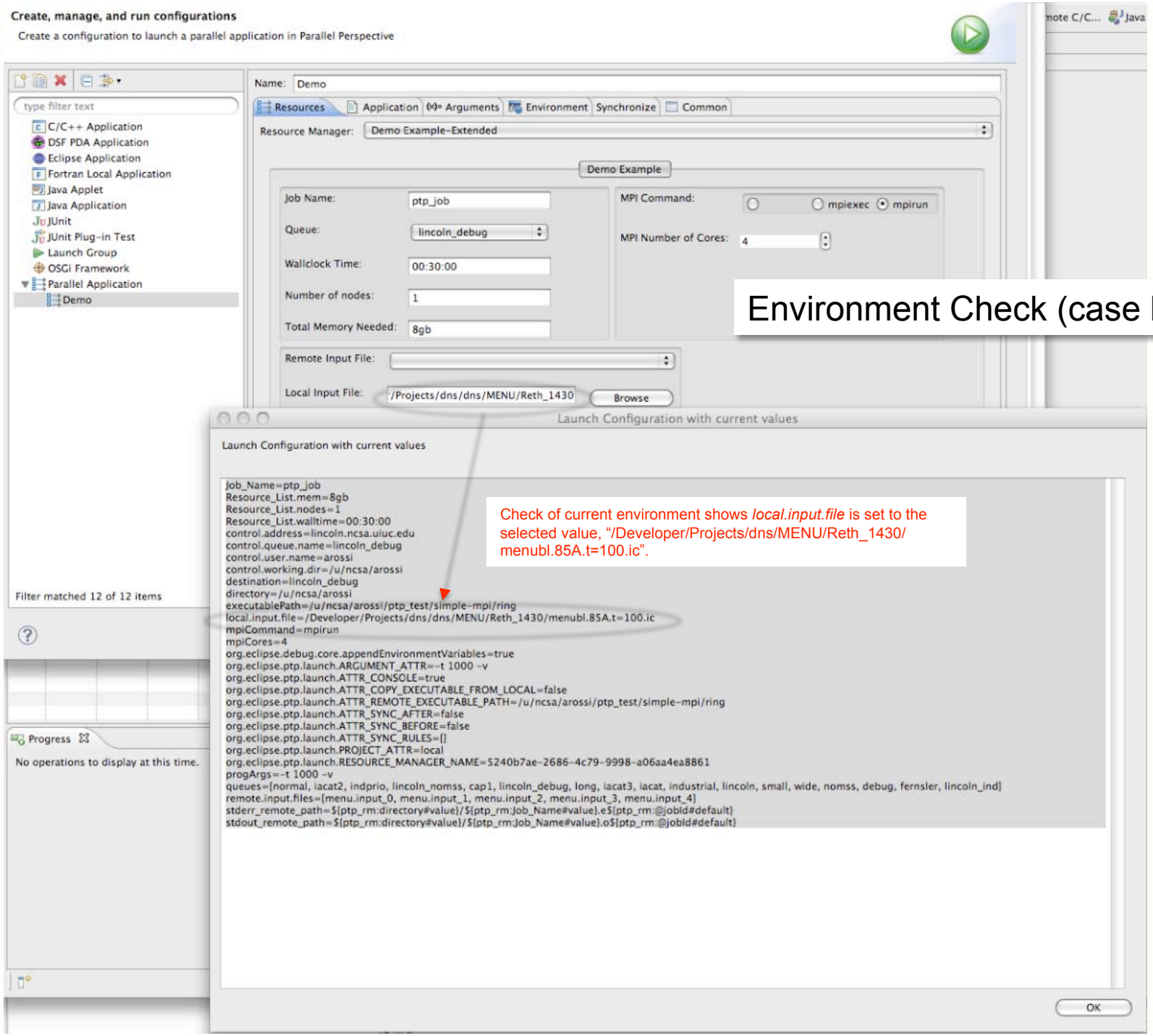
Setting *resolve* to false tells the Resource Manager not to interpret the argument (otherwise it would think the shell variable ${INPUT} was an Eclipse variable).

```
<line>
    <arg resolve="false">if [ -n "${COMMAND}" ] ; then</arg>
</line>
<line>
    <arg resolve="false"> COMMAND="${COMMAND} ${MPI_ARGS}</arg>
    <arg>${ptp_rm:executablePath#value} ${ptp_rm:progArgs#value}</arg>
    <arg resolve="false">${INPUT}"</arg>
</line>
<line>
    <arg resolve="false">else</arg>
</line>
<line>
    <arg> COMMAND="${ptp_rm:executablePath#value} ${ptp_rm:progArgs#value}</arg>
    <arg resolve="false">${INPUT}"</arg>
</line>
<line>
    <arg resolve="false">fi</arg>
</line>
```

Environment Check (case a)



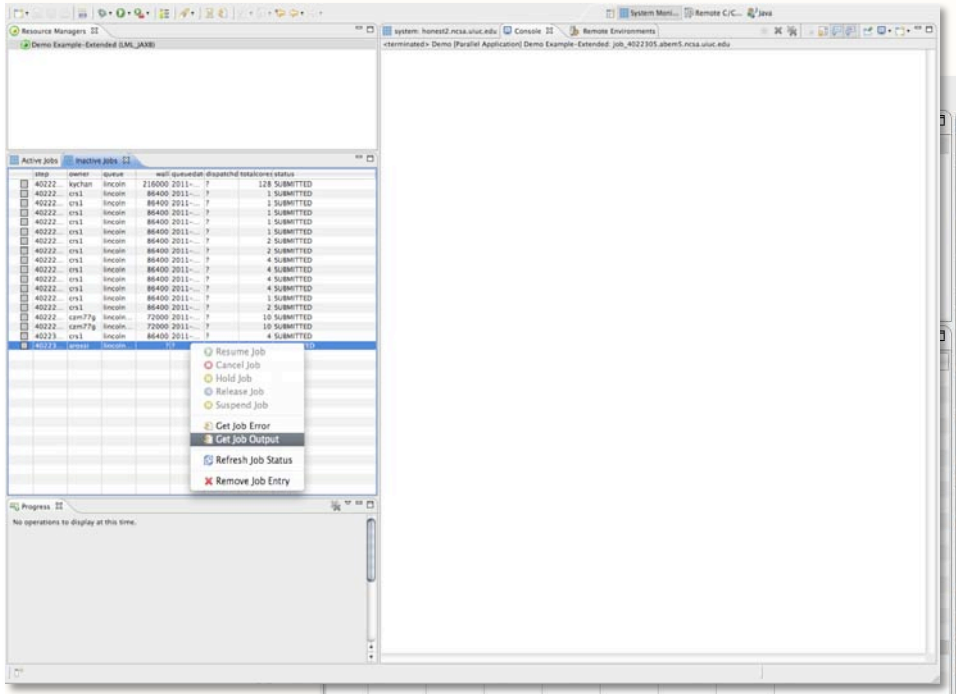
Environment Check (case b)



Check of current environment shows *local.input.file* is set to the selected value, “/Developer/Projects/dns/MENU/Reth_1430/menubl.85A.t=100.ic”.

Run Result (case b)

Running from the last setting (*local.input.file*); output of job shows remote file to be correct (local file name in the *.eclipsesettings* directory).



System Moni... Remote C/C... Java

system: honest2.ncsa.uiuc.edu | Console | Remote Environments

```
/u/ncsa/arossi/ptp_job.o4022305
/dev/sda2 on /tmp type ext2 (rw)
----------------------------------------
Begin Torque Prologue (Mon Jun 13 09:56:44 2011)
Job ID:          4022305
Username:        arossi
Group:           adn
Job Name:        ptp_job
Limits:          mem=8gb,ncpus=1,neednodes=1,nodes=1,walltime=00:30:00
Job Queue:       lincoln_debug
Account:         lincoln_adn
Nodes:           abe1208
End Torque Prologue
----------------------------------------
Warning: no access to tty (Bad file descriptor).
Thus no job control in this shell.
fake input file: .eclipsesettings/menubl.85A.t=100.ic
my_id 1 numprocs 4
Slave 1: top of trip 1 of 1000: before receiving from source=0
fake input file: .eclipsesettings/menubl.85A.t=100.ic
my_id 0 numprocs 4
Master: starting trip 1 of 1000: before sending num=1 to dest=1
Slave 1: inside trip 1 of 1000: after receiving passed_num=1 from source=0
Slave 1: inside trip 1 of 1000: before sending passed_num=2 to dest=2
Master: inside trip 1 of 1000: before receiving from source=3
Slave 1: bottom of trip 1 of 1000: after send to dest=2
Slave 1: top of trip 2 of 1000: before receiving from source=0
fake input file: .eclipsesettings/menubl.85A.t=100.ic
my_id 3 numprocs 4
Slave 3: top of trip 1 of 1000: before receiving from source=2
Master: end of trip 1 of 1000: after receiving passed_num=4 (should be =trip*numprocs=4) from source=3
Master: starting trip 2 of 1000: before sending num=5 to dest=1
Master: inside trip 2 of 1000: before receiving from source=3
Master: end of trip 2 of 1000: after receiving passed_num=8 (should be =trip*numprocs=8) from source=3
fake input file: .eclipsesettings/menubl.85A.t=100.ic
my_id 2 numprocs 4
Slave 2: top of trip 1 of 1000: before receiving from source=1
Slave 2: inside trip 1 of 1000: after receiving passed_num=2 from source=1
Slave 2: inside trip 1 of 1000: before sending passed_num=3 to dest=3
Slave 2: bottom of trip 1 of 1000: after send to dest=3
Slave 2: top of trip 2 of 1000: before receiving from source=1
Slave 2: inside trip 2 of 1000: after receiving passed_num=6 from source=1
Slave 2: inside trip 2 of 1000: before sending passed_num=7 to dest=3
Slave 2: bottom of trip 2 of 1000: after send to dest=3
Slave 2: top of trip 3 of 1000: before receiving from source=1
Slave 2: inside trip 3 of 1000: after receiving passed_num=10 from source=1
Slave 1: inside trip 2 of 1000: after receiving passed_num=5 from source=0
Slave 1: inside trip 2 of 1000: before sending passed_num=6 to dest=2
Slave 1: bottom of trip 2 of 1000: after send to dest=2
Slave 1: top of trip 3 of 1000: before receiving from source=0
Slave 1: inside trip 3 of 1000: after receiving passed_num=9 from source=0
Slave 1: inside trip 3 of 1000: before sending passed_num=10 to dest=2
Slave 1: bottom of trip 3 of 1000: after send to dest=2
Slave 1: top of trip 4 of 1000: before receiving from source=0
Slave 3: inside trip 1 of 1000: after receiving passed_num=3 from source=2
Slave 3: inside trip 1 of 1000: before sending passed_num=4 to dest=0
Slave 3: bottom of trip 1 of 1000: after send to dest=0
Slave 3: top of trip 2 of 1000: before receiving from source=2
Slave 3: inside trip 2 of 1000: after receiving passed_num=7 from source=2
Slave 3: inside trip 2 of 1000: before sending passed_num=8 to dest=0
Slave 3: bottom of trip 2 of 1000: after send to dest=0
Slave 3: top of trip 3 of 1000: before receiving from source=2
Master: starting trip 3 of 1000: before sending num=9 to dest=1
Master: inside trip 3 of 1000: before receiving from source=3
```

Progress

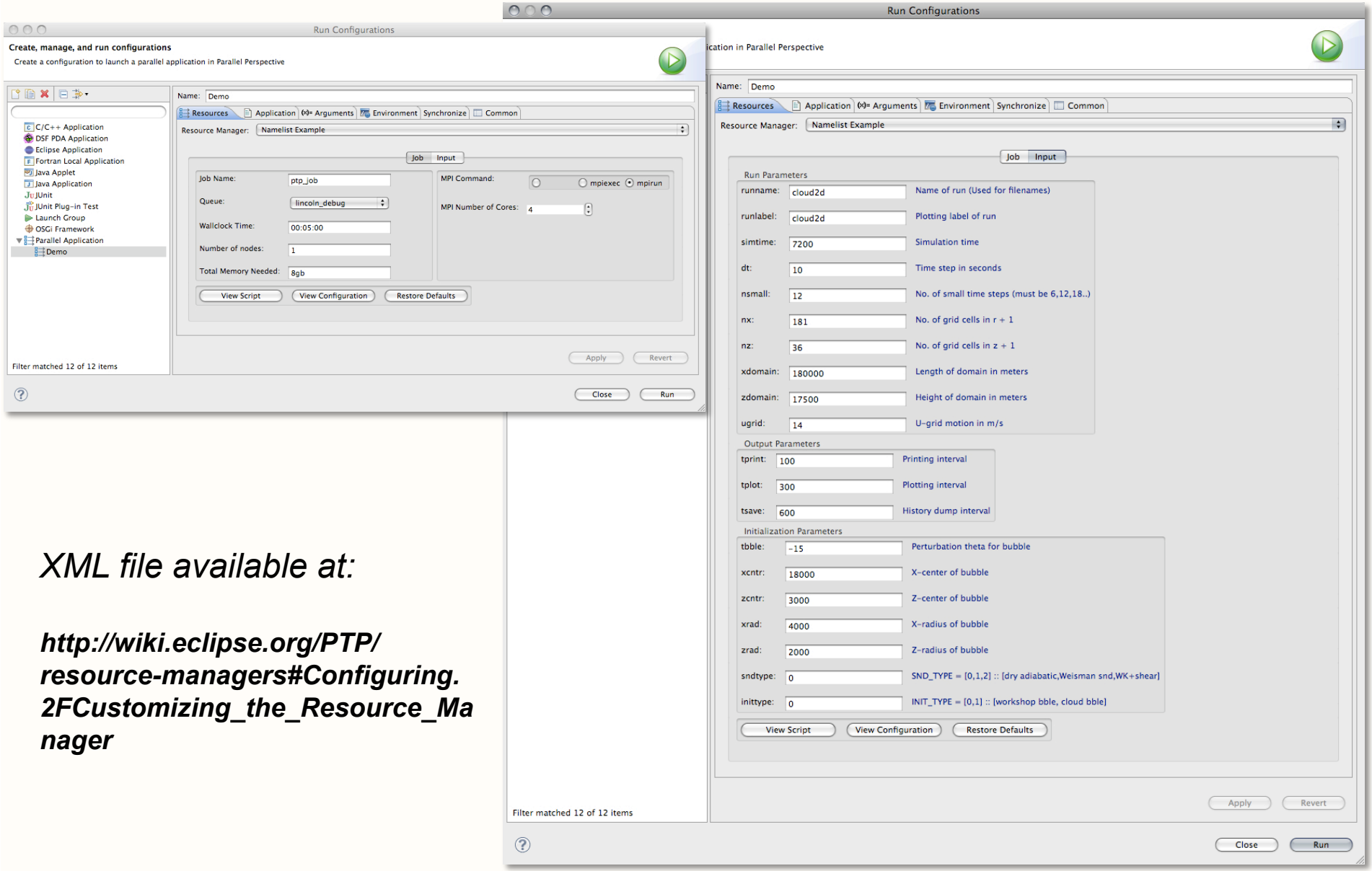
No operations to display at this time.

Final Example: Configurable Input

As a final example of some of the possibilities afforded by the Configurable Resource Manager, we show here two screen shots of a dual-panel Launch Tab, the second of which allows the user to configure the values in an input parameter (Fortran “namelist”) file which is staged (as in the preceding example) over as a managed file.

This example required the addition of attributes corresponding to the variables, along with default values, specified by the input file, the addition of three composites of text widgets, and the specification of a managed file whose content resembles that of the <script> element: a series of <line> elements containing resolvable <arg> elements.

Configurable Job Input



XML file available at:

http://wiki.eclipse.org/PTP/resource-managers#Configuring.2FCustomizing_the_Resource_Manager

Additional/Fixed in *PTP* 5.0.1

1. Bug fix for 0-length path of managed file
2. Bug fix for **working directory** of <command> (was not fully implemented)
3. Generalized line-argument content for managed file (namelist input tab example)
4. <button-action>: commands (which do not reference *@jobId*) can be run from the Launch Tab via a push-button