NOAO SDM Document PL002

The NHPPS Pipeline Definition Language (PDL)

F. Valdes[1], F. Pierfederici[1,2], D. Scott[1]

National Optical Astronomy Observatories
Science Data Management

V2.0: February 9, 2012

[1]NOAO Science Data Management, P.O. Box 26732, Tucson, AZ 85732
[2]Currently: Space Telescope Science Institute, 3700 San Martin Dr., Baltimore, MD 21218

Table of Contents

List of Figures

1 Introduction

The *NOAO High Performance Pipeline System* (NHPPS) [1] orchestrates *pipelines*. A pipeline consists of a set of steps or stages called *modules*. The modules are *triggered* by *events* which the it is looking for such as the appearance of a file in a directory or a change in a *blackboard*. When a module is triggered it performs a set of *actions*; a set preprocessing actions, a processing action, and a set of final actions. Different sets of final actions are selected depending on the *exit status* of the processing action. For more complete details on NHPPS see reference [1].

The definition of the pipelines, modules, and the various actions of a module are specified by the NHPPS *Pipeline Definition Language* (PDL). The general features and the technical specification of the PDL is the subject of this document. This document describes the V2.0 PDL which is an extension of the previous version.

The NHPPS PDL provides a 3 level inheritance hierarchy for commands an options, essentially XML elements and attributes. These levels are **System**, **Pipeline**, and **Module**. This tri-layered approach allows elements to be defined globally for all pipelines forming a *Pipeline Application*, globally for all modules in a pipeline, or individually for a module. In this hierarchy the elements defined at a more global level are inherited only if the same element is not defined at the lower level.

The PDL is expressed in the *eXtensible Markup Language* (XML). The files defining an NHPPS pipeline application consist of a single system XML file and separate pipeline XML files. The system file has only global elements while the pipeline files have global elements and module elements.

Compiling the PDL files means merging the system level global elements into the pipeline global elements and then merging the pipeline global elements into the module elements of the pipeline. This compiling takes place when a *Pipeline Manager* is invoked to orchestrate a pipeline. The merged PDL is written to the pipeline manager's standard output for reference. Normally the PDL source is left in the layered form but the NHPPS command `pdlcompile` may be used to generate the fully merged pipeline PDL version. This compiled version is mostly intended for checking the correctness of the inheritance definitions, though it is legal to use the compiled version in place of the layered version.

Two important new features in V2.0 not found in the earlier version are a *templating* mechanism for modules and a mapping of exit status codes and to *logical identifiers*. Because it is typical that modules have very similar definitions, both within a pipeline and across all pipelines in an application, the templating allows much simpler pipeline PDL descriptions and also enforcing standards across the pipelines in an application. The use of logical names for processing command exit codes, with associated attributes, such as severity levels, also improves PDL readability and standardization within an application.

The general and technical descriptions in this document can be initially difficult to grasp. Reference to the example in the appendix can be a good aid to understanding the description.

2 The Element Hierarchy

The PDL elements which may appear only in the System and Pipeline global levels are: **ExitCodes** and **Template**. The elements which may appear in all three levels are: **Directories**, **MaxExec**, **MinDisk**, **Trigger**, **PreProcAction**, **ProcAction**, **PostProcAction**. The the attribute **maxActive** may also appear in the three levels. The **Module** element only appears in the pipeline PDL.

The **Template** element is used, at the system and pipeline levels, to group inherited values for inclusion only in modules that identify themselves as of that template type.

3 System PDL File

The System PDL file is used to define inheritable elements for the ALL of the pipelines, and subsequently the pipelines modules, which are members of the System. The root element of a System PDL file is the **System** element.

4 Pipeline PDL Files

The Pipeline PDL file defines default elements for the modules within the pipeline. Additionally, and more importantly, the Pipeline PDL file defines the *Modules* which comprise the pipeline. The root element of a Pipeline PDL file is the **Pipeline** element.

5 Macros

To provide a level of flexibility in the PDL files, 3 Macros have been defined. Macros are executed when the XML file is read.

ENV: This macro replaces its argument with the value of the environment variable with the same name as the argument. For example if `$PIPENAME` has the value 'dir' then, `<Pipeline name="ENV(PIPENAME)">` would be interpreted as `<Pipeline name="dir">`.

Do not use the ENV macro on environment variables you want to have interpreted during runtime (variables that may change), as the ENV macro, like all the others, is interpreted when the PDL file is read.

CAPS: This macro replaces its argument with a completely capitalized version. For example, `x="CAPS(dir)DATA"` is interpreted as `x="DIRDATA"`.

INDEX: This macro replaces its argument (the name of a **Module** in the **Pipeline**) with the index of the **Module**.

6 Variable Substitution

The PDL parser has the ability to replace variables which have the syntax `${...}`. In particular, attributes of elements may be specified as `${element.attribute}`. The special element, `self` refers to the **Module** in which the variable exists, otherwise the tree from the root PDL element to the attribute must be specified.

The most widely used variable is `${self.name}` which may be shortened to `${name}`. The variable `${pipe}` is similarly a short form for name attribute of the Pipeline element (i.e. `${Pipeline.name}`).

An attribute which is important in allowing module templates to be described and then substituted in specific modules is `vars`. This attribute of the **Module** element has a value consisting of a string of words; e.g. `vars="a b c"`. The full string may be substituted in elements within a module by `$vars` for the whole string and `${vars[1]}`, ${vars[2]}, etc. for the words. Note this is one-indexed.

7 Exit Codes

NHPPS executes commands specified in the action elements. These commands, which may be build-in, plug-in, or host, return an exit status. The PDL, and NHPPS, is designed around responding to these exit status in various ways. Due to the nature of host commands which only return an integer exit code (in the range 0-255) it is desirable to map these codes to logical identifiers with associated attributes. This is done using the **ExitCodes** element.

This element associates a *code* with a logical *id*, a category *type*, and a *desc*ription. In addition, a blackboard *flag* (a single character) may be associated with the code. The code and id must be unique. When an association is defined the logical id may be used in the PDL instead of a code.

The set of associations forms a look-up table or map. This information is compiled and stored at the same time as the PDL is compiled by the Pipeline Manager. This stored map can then be used with the `osf_exitinfo` command in the host commands to map an id in the program to an exit code which is then mapped back to the id in NHPPS for action in the PDL logic. This is allows developers to think purely in terms of logical ids.

The category types are useful in association with tools that query the blackboard (where the exit ids are also stored) and the compiled mapping to react to classes of module results. These classes are basically severity levels from complete success, to recoverable errors, to fatal errors.

The flag values are only needed for `osf_exitinfo` to interpret blackboard flags that are set independently of an action status code returned by an action and subsequently entered into the blackboard. For instance, if the blackboard is updated by an `osf_update` command in the PDL or from the command line.

Because the information is part of the PDL, and compiled when a pipeline is started, it ensures the consistency of the mappings if the PDL is changed. The exit code elements, which occur only in the system and pipeline global definitions, allow defining pipeline application standards in the System PDL as well as more specialized exit ids in specific pipelines.

Note that if there are no **ExitCodes** elements the PDL can still be used but the identifier strings

will simply be the integer codes; i.e. if there is no mapping for an exit code to an id, the exit code becomes the id.

8 Templates

The V2.0 PDL provides a templating system for modules. A template is very much like a module in that can contain all the same elements. However, it is uses variable substitution to apply to various modules. The templates can be either at the system level or the pipeline level. Since most pipelines in a pipeline application typically have similar types of modules, for instance for initializing or finishing up or just a simple blackboard trigger running a host command of the same name as the module, it makes sense to abstract these as system level templates.

The templates and modules have a **type** attribute which is used to associate a template with a module. The value of the types is completely up to the developer. When a matching type between a module and a template is found the elements of the template are merged into the module definition. If a module does not specify a type the default is a type of ”Module”. The rules are:

1. **MaxExec**, **MinDisk**, **PreProcAction**, and **ProcAction**, where there can only be one element per module, the template element is only added if it is not present in the module definition.
2. **Trigger**, which may have multiple instances in a module, will only be added if there are no trigger elements in the module; i.e. there is no merging.
3. **PostProcAction**, which generally have multiple instances in a module, are matched by the **val** attribute of the **ExitCode** element they contain. Only template PostProcAction elements with exit codes which don’t match an exit code in the module are added.
4. **MaxExec** and **MinDisk**, which can be defined in four places, have the following precedence: module, pipeline global, template, system global.

As noted in the introduction, the templating mechanism allows much more concise and consistent pipeline descriptions. This addresses comments sometimes made about the verboseness of XML syntax used to express the PDL. Of course, while the many pipeline PDL files in an application can be made more compact and readable, the complexity is moved into construction of the templates.

References

[1] F. Valdes, T. Cline, F. Pierfederici, B. Thomas, M. Miller, and R. Swaters. The NOAO High-Performance Pipeline System. SDM Pipeline Document PL001, NOAO/SDM, Oct 2006. http://chive.tuc.noao.edu/noaodpp/Pipeline/PL001.pdf.

Appendix A: PDL Technical Specification

The technical specification of the XML used by the Pipeline Definition Language is defined by the *EaseXML* package used to handle the files. With this package the effective DTD is defined in `$NHPPS/src/python/nhpps/dtd/dtd.py`. The elements of the PDL specification are summarize below.

PDL Elements

Child elements are denoted as required (no marking, ex: **Description**), 0 or 1 marked with ? (ex: **MaxExec**?), 0 or more marked with * (ex: **PostProcAction***), and 1 or more marked with + (ex: **Module**+). Attributes can be required or optional as indicated by `[]`. If there is no `/` indicated, because there are child elements, then a matching closing tag is required.

`<System>`

The **System** element is used as the root element of the system level PDL file. The elements in the System element provide default values in the event they are not defined in the **Pipeline** or **Module** elements. This provides the pipeline architect with a method of defining elements to be used throughout all of the pipelines without needing to duplicate the definitions in each Pipeline PDL file. This also has the effect of providing conventions for all pipelines in the system. In particular, the **ExitCodes** elements define conventions for module exit codes and the **Template** elements provide templates for common types of types of modules.

children: **Description**, **Directories**, **MaxExec**?, **MinDisk**?, **ExitCodes***, **Template***

`<Pipeline system="" name="" poll="" [maxActive=""]>`

The **Pipeline** element is the root of a Pipeline PDL file. Values which are not defined at this level are taken from the **System** element. Each 'pipeline' within the pipeline application has an PDL file which declares a single pipeline element.

children: **Description**, **Directories**?, **MaxExec**?, **MinDisk**?, **ExitCodes***, **Template***, **Module**+

`<Module name="" [type=""] [var=""] [maxActive=""] [isActive=""] [/]>`

The **Module** element defines the actions a module executes. Missing children are supplied from the **Pipeline** or **System** level or DTD defaults.

children: **Description**?, **Directories**?, **Trigger***, **MaxExec**?, **MinDisk**?, **PreProcAction**?, **ProcAction**?, **PostProcAction***

`<Description>`

The **Description** tag provides a region in which to explain the purpose of the parent element. This allows for a self-documenting programming style (if filled with meaningful information).

children: Any text description. This may be in restructured text syntax which can be used by formatting tools such as `plman`.

`<Directories trig="" data="" input=""/>`

The Directories element is used to specify the locations of the directories the pipeline should use to store its data, read input, etc... These are usually specified using environment substitution.

`<MaxExec time="" status="">`

The **MaxExec** element provides a means to constrain the amount of time that a **Module**'s ProcAction is allowed to execute before being killed as a hung process. The status attribute is the exit code that is normally handled by a **PostProcAction** action. The time is specified in the form days:hours:minutes:seconds (e.g. 0:1:0:0 is one hour).

`<MinDisk space="" status="">`

The MinDisk tag is used to specify the minimum amount of disk space required in order for a **Module**'s ProcAction to be executed, as well as an exit code to return in the event there isn't enough disk space available.

`<ExitCodes id="" code="" severity="" [flag=""] [desc=""]/>`

This tag associates an exit code with a logical identifier and associated attributes. The identifiers and codes must be unique across all elements. The set of ExitCodes elements form a mapping used in the PDL and the blackboard.

`<Template type="">`

The Template element provides a container for default **Trigger**, **PreProcAction**, **ProcAction**, **PostProcAction**, **MaxExec**, and **MinDisk** to be applied to modules that don't define these elements. Templates have a **type** which is used provide multiple templates of different types. A module is associated with a template by the type.

children: **Description**?, **Trigger**?, **PreProcAction**?, **ProcAction**?, **PostProcAction**?, **MaxExec**?, **MinExec**?

`<PreProcAction>`

A **PreProcAction** describes the actions that are to occur when a **Module** is executed, in order to prepare for the **Module**'s **ProcAction**. There may only be one PreProcAction per **Module**, however a **PreProcAction** may have several commands.

children: (**Foreign** | **PlugIn** | **OSFUpdate** | **OSFUpdateParent** | **OSFWait** | **OSFClose** | **OSFConditRemove** | **OSFFinal**| **OSFOpen** | **RenameTrigger** | **RemoveTrigger**)+

`<ProcAction>`

The ProcAction element specifies the primary action the **Module** is responsible for. There may only be a single action.

children: (**Foreign** | **PlugIn** | **OSFUpdate** | **OSFUpdateParent** | **OSFWait** | **OSFClose** | **OSFConditRemove** | **OSFFinal**| **OSFOpen** | **RenameTrigger** | **RemoveTrigger**)+

`<PostProcAction>`

The PostProcAction describes the actions a **Module** should take after the **ProcAction** , based upon the value of the **ExitCode** from the **ProcAction**. There may be several PostProcActions in a **Module**, to specify different actions for different types of results from the **ProcAction**.

children: **ExitCode**+, (**Foreign** | **PlugIn** | **OSFUpdate** | **OSFUpdateParent** | **OSFWait** | **OSFClose** | **OSFConditRemove** | **OSFFinal**| **OSFOpen** | **RenameTrigger** | **RemoveTrigger**)+

`<Foreign argv=""/>`

The Foreign element provides a pipeline architect the ability to launch programs as a valid action within the pipeline. This allows the pipeline to call any program on the system allowing flexibility in the design/coding of the scientific modules which compose the core code of the pipeline. **argv** specifies the name of the command and any arguments.

`<PlugIn argv="">`

The PlugIn element provides the capability to call functions in the NHPPS_PlugIns/actions.py file. **argv** specifies the name of the function and any arguments needed.

`<OSFUpdate argv=""/>`

The OSFUpdate element allows the pipeline system to update the OSF blackboard of the current dataset. The **argv** specifies the module name or index and the blackboard value to set. The value has the form `[char|.][:status]` where char is a character, the period specifies the flag value is to be unchanged, and an integer "status" value may follow using a colon delimiter.

`<OSFUpdateParent argv=""/>`

The OSFUpdateParent element is like OSFUpdate except the dataset name is first modified to trim the from the last hyphen. For example, a dataset of "a_b-c-d_e-f" would be trimmed to "a_b-c-d_e". This is a special used that depends on the dataset naming convention of the pipeline application and is used when file triggers are received from daughter pipelines where the file name is the daughter dataset name. The **argv** is the same as for OSFUpdate.

`<OSFWait argv=""/>`

The OSFWait element is a combination of a conditional blackboard update and a decrement counter in the counter field of a blackboard entry. When used on a module entry where the counter has not be set it the flag argument is "flag:counter" (e.g. `w:$NCALL`) which sets the flag field to the specified value and the counter to the specified integer value. Typically the value is the environment variable NCALL set by the status return of a module. Subsequent calls are of the form "flag". Each time the action is called the counter field is decremented by

one without changing the current flag value. When the counter reaches zero the flag value is set. This action is used to implement a map/reduce strategy.

`<OSFClose/>`

Closes the dataset on which the **Pipeline** was processing. The dataset is not removed from the blackboard.

`<OSFOpen/>`

Opens the dataset, in case it was closed, which the **Pipeline** is trying to process.

`<OSFFinal/>`

The dataset blackboard entry is closed, in not done previously, and then removes it from the blackboard.

`<OSFConditRemove/>`

The dataset blackboard entry is closed, if not done previously, but only removed from the blackboard if the flags satisfy a pattern. Currently the pattern is fixed to be a 'c' in the first stage, any combination of 'c', 'n', or '_' in the following stages, and ending with 'd'. In other words, it will not remove blackboard entries that have other flags which likely signify a error of some kind.

`<RenameTrigger argv=""/>`

Renames the trigger file which is specified in **argv** to a new name, also specified in **argv**.

`<RemoveTrigger argv=""/>`

Removes the trigger file specified in **argv**.

`<ExitCode val=""/>`

Specifies an exit code which causes the actions in a **PostProcAction** to be executed. The *val* is that is matched against the current exit status from the ProcAction action or from MaxExec, or MinDisk check.

`<Trigger conditional="">`

Specifies the required events which must occur before a **Module** will begin execution. The conditional may be "AND" or "OR" for combining multiple requirement children. Note that there may be multiple Trigger elements in a module and the module is triggered if any of then is satisfied.

children: (**FileRequirement** | **OSFRequirement** | **TimeRequirement**)+

`<FileRequirement directory="" fnPattern=""/>`

Defines an event which will trigger when a file matching **fnPattern** is found in **directory**. Typically the file name pattern will be generated with wildcards and environment variables for the file event being checked.

`<OSFRequirement argv="">`

Defines a requirement which triggers when the status of a **Module**, named in **argv**, relates to a value, also in **argv**, in a manner specified in **argv**. For example ”dirstart = x” would cause the trigger to go off if status of module dirstart is ’x’.

`<TimeRequirement start="" end="" interval=""/>`

Defines a trigger which occurs at a specified **interval**, between **start** and **end**.

PDL Attributes

The following element attributes are defined within select NHPPS PDL elements. If no default is shown then there is no default. The Boolean datatype may take any of the the values ”True”, ”T”, ”true”, ”t”, ”Yes”, ”Y”, ”yes”, ”y”, ”False”, ”F”, ”false”, ”f”, ”No”, ”N”, ”no”, ”n”.

maxActive

If greater than 0, this specifies the maximum number of open OSF Datasets to allow at a time; otherwise it is ignored. This is typically used to restrict only a single instance of a module to run to avoid conditions where multiple instances might interfere with each other.

required: No, datatype: Integer, default: 0

system

Defines the system name for the pipeline system. This value is used to find the system level PDL file. For example, if system=’Mario’ then the System level XML file is Mario.xml.

required: Yes, datatype: String

name

Specifies the name of either a **Pipeline** or a **Module**.

required: Yes, datatype: String

poll

Specifies the polling time, or amount of time a **Module** sleeps between checking for triggering events, in fractions of a second.

required: Yes, datatype: Float

root

Defines the directory which acts as the root in which the pipeline writes files.

required: No, datatype: String, default:
`ENV($NHPPS_DATA)/CAPS($Pipeline.system_$Pipeline.name)`

input

Defines the directory from which the pipeline gets its input data.

required: No, datatype: String, default: `$Pipeline.Directories.root/input`

obs

Specifies the directory into which the OSF Blackboard entries are written.

required: No, datatype: String, default: `$Pipeline.Directories.root/obs`

error

Reason unknown, its value is used as the NHPPS_DIR_ERROR environment variable.

required: No, datatype: String: default: `$Pipeline.Directories.root/err`

time

Sets the maximum amount of time a **Module**'s ProcAction has available to execute before being killed as a hung process.

required: Yes, datatype: String, default: `0:0:0:0`

The format is D:H:M:S where D is the number of days, H is the number of hours, M the number of minutes, and S the number of seconds.

status

Defines the exit code value to use, rather than an actual exit code, in the event a **Module**'s ProcAction does not execute or is killed due to either insufficient disk space or it exceeded its time limit.

required: No, datatype: String, default: `0`

space

Defines the minimum amount of disk space that must be available in order for the **Module**'s ProcAction to execute.

required: Yes, datatype: String, default: 0

The format may include an expression for Python to evaluate or may end in k,K,m,M,g, or G to represent the space is in kilo, mega, or giga bytes. When the PDL is compiled the space is always converted to k.

isFailure

Specifies that the **ExitCode**s contained within the **PostProcAction** represent **ExitCode**s of error or failure conditions.

required: No, datatype: Boolean, default: False

isActive

Specifies whether or not the **Module** is active. If it is, it is in included in the pipeline PDL tree, otherwise it is ignored.

required: No, datatype: Boolean, default: True

argv

Provides needed input to several commands.

required: Yes, datatype String

val

Defines an exit code that the **PostProcAction** will respond to.

required: Yes, datatype: String, default: `_` or `default`

conditional

Specifies whether all of the requirements within the **Trigger** element are required (”AND”) or if any of them will do (”OR”). Those are the only values allowed.

required: No, datatype: String, default: `AND`

directory

Specifies the directory in which the **FileTrigger** looks for files which match **fnPattern**.

required: No, datatype: String, default: `$Pipeline.Directories.input`

fnPattern

Specifies the search pattern to use within the **FileTrigger** when looking for trigger files.

required: Yes, datatype: String

start

Provides the starting date and time, meaning the event will not trigger until this point has past, for a **TimeRequirement**.

required: No, datatype: String, default: Current Time

The format is ”YYYY-MM-DDThh:mm:ss”, where YYYY is the year (*must* be 4 digits), MM is the month, DD is the day, hh is the hour, mm is the minute, ss is the second.

end

Provides the end date and time, meaning the event will not trigger after this point, for a **TimeRequirement**.

required: No, datatype: String, default: `9999-12-31T23:59:59`

The format is ”YYYY-MM-DDThh:mm:ss”, where YYYY is the year (*must* be 4 digits), MM is the month, DD is the day, hh is the hour, mm is the minute, ss is the second.

interval

Provides the interval at which the **TimeRequirement** will trigger between **start** and **end**.

required: Yes, datatype: String

The format is ”D:H:M:S”, where D is the number of days, H the number of hours, M, the number of minutes, and S is the number of seconds between triggering events.

id

The logical ID for an exit code. This must be unique across all **ExitCodes** elements.

required: Yes, datatype: String

code

The integer exit code from a process exit to be mapped to a logical ID. This must be unique across all **ExitCodes** elements.

required: Yes, datatype: Integer

flag

A string of flag values to be used to interpret a blackboard entry where a module has a flag value without an associated status code. If any of the characters in the flag value match a blackboard flag the logical ID is associated with the module. The characters must be unique across all **ExitCodes** elements.

required: No, datatype: String

severity

A severity ID for the status ID. This need not be unique for the **ExitCode** elements.

required: Yes, datatype: String

desc

A description of the status.

required: No, datatype: String

type

An identifier used in the **Template** and **Module** elements to match them.

Template: required: Yes, datatype: String

Module: required: No, datatype: String, default: Module

Appendix B: Example

The following is a very simplified example. Figure 1 shows a pipeline PDL file demonstrating how compact the description can be with the use of templates. In this example there are two standard modules where the first triggers on a file and the second triggers on the blackboard when when the first completes.

Figure 1: Example pipeline PDL file.

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE Pipeline SYSTEM "NHPPS.dtd">
<Pipeline system="ENV(NHPPS_SYS_NAME)" name="ex" poll="0.1">

<Description> Simple example.  </Description>
<Module name="exstart" type="StartPipe"/>
<Module name="exdone" type="DonePipe" var="exstart"/>

</Pipeline>
```

An example system PDL file is shown in figure 2. This contains some exit code mappings, a system value of timeout, and two templates for the types in the pipeline of figure 1.

Figure 3 is the compiling of the above files, using `pdlcompile`. This shows the global **MaxExec** merging, the addition of a default for **MinDisk**, the merging of the template contents, and the variable substitutions from environment variables, PDL elements, and the var attribute.

Figure 2: Example system PDL file.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE System SYSTEM "NHPPS.dtd">
<System>
<Description> Example System </Description>

<ExitCodes id="PROCESSING" code="2" type="OK"    desc="normal processing"/>
<ExitCodes id="COMPLETED"  code="3" type="OK"    desc="normal completion"/>
<ExitCodes id="FATAL"      code="4" type="FATAL" desc="fatal error"/>
<ExitCodes id="TIMEOUT"    code="5" type="HALT"  desc="module timeout"/>

<!-- Set 'global' maximum execution time to 1 hour (day:hr:min:sec)   -->
<MaxExec time="0:1:0:0" status="TIMEOUT"/>

<Template type="StartPipe">
  <Description> Defaults for StartPipe modules.  </Description>
  <Trigger>
    <FileRequirement fnPattern="*.$pipetrig"/>
  </Trigger>
  <PreProcAction>
    <OSFUpdate argv="$pipestart w"/>
    <RenameTrigger argv="$EVENT_NAME.$pipetrig $EVENT_NAME.$pipeproc"/>
  </PreProcAction>
  <ProcAction>
    <Foreign argv="StartPipe"/>
  </ProcAction>
  <PostProcAction>
    <ExitCode val="COMPLETED"/>
    <OSFUpdate argv="$pipestart c"/>
    <RemoveTrigger argv="$EVENT_NAME.$pipeproc"/>
  </PostProcAction>
  <PostProcAction>
    <ExitCode val="default"/>
    <OSFUpdate argv="$name f"/>
    <RenameTrigger argv="$EVENT_NAME.$pipeerr"/>
  </PostProcAction>
</Template>

<Template type="DonePipe">
  <Description> Defaults for DonePipe modules.  </Description>
  <Trigger>
    <OSFRequirement argv="$var[1] == c"/>
    <OSFRequirement argv="$name == _"/>
  </Trigger>
  <PreProcAction>
    <OSFUpdate argv="$name p"/>
  </PreProcAction>
  <ProcAction>
    <Foreign argv="DonePipe"/>
  </ProcAction>
  <PostProcAction>
    <ExitCode val="COMPLETED"/>
    <OSFUpdate argv="$name d"/>
  </PostProcAction>
  <PostProcAction>
    <ExitCode val="default"/>
    <OSFUpdate argv="$name f"/>
  </PostProcAction>
</Template>

</System>
```

Figure 3: Compiled PDL from the source files in figures 1 and 2.

```
<Pipeline system="ExampleSystem" name="ex" poll="0.10" maxActive="0">
  <Description> Simple example.  </Description>
  <ExitCodes id="COMPLETED" code="3" type="OK" desc="normal completion"/>
  <ExitCodes id="FATAL" code="4" type="FATAL" desc="fatal error"/>
  <ExitCodes id="TIMEOUT" code="5" type="HALT" desc="module timeout"/>
  <Module name="exstart" type="StartPipe" maxActive="0" isActive="True">
    <Trigger conditional="AND">
      <FileRequirement directory="example/trigger" fnPattern="*.extrig"/>
    </Trigger>
    <MaxExec time="0:1:0:0" status="TIMEOUT"/>
    <MinDisk space="0k" status="0"/>
    <PreProcAction>
      <OSFUpdate argv="exstart w"/>
      <RenameTrigger argv="$EVENT_NAME.extrig $EVENT_NAME.exproc"/>
    </PreProcAction>
    <ProcAction>
      <Foreign argv="StartPipe"/>
    </ProcAction>
    <PostProcAction>
      <ExitCode val="COMPLETED"/>
      <OSFUpdate argv="exstart c"/>
      <RemoveTrigger argv="$EVENT_NAME.exproc"/>
    </PostProcAction>
    <PostProcAction>
      <ExitCode val="default"/>
      <OSFUpdate argv="exstart f"/>
      <RenameTrigger argv="$EVENT_NAME.exerr"/>
    </PostProcAction>
  </Module>
  <Module name="exdone" type="DonePipe" maxActive="0" isActive="True">
    <Trigger conditional="AND">
      <OSFRequirement argv="exstart == c"/>
      <OSFRequirement argv="exdone == _"/>
    </Trigger>
    <MaxExec time="0:1:0:0" status="TIMEOUT"/>
    <MinDisk space="0k" status="0"/>
    <PreProcAction>
      <OSFUpdate argv="exdone p"/>
    </PreProcAction>
    <ProcAction>
      <Foreign argv="DonePipe"/>
    </ProcAction>
    <PostProcAction>
      <ExitCode val="COMPLETED"/>
      <OSFUpdate argv="exdone d"/>
    </PostProcAction>
    <PostProcAction>
      <ExitCode val="default"/>
      <OSFUpdate argv="exdone f"/>
    </PostProcAction>
  </Module>
</Pipeline>
```