

# Scalable System Software Process Manager

## Specification Draft

May 6, 2004

## 1 Functionality

The process manager provides the ability to start and control parallel process groups across parallel machines. It provides parallel process group bootstrapping, allowing parallel processes (like MPI programs) to execute properly. It also allows complete specification of different executables, arguments and environment variables across different ranks. Resource limits supported by *rlimit* are supported as startup specifications. This functionality is provided by the *create-process-group* command. Once a process group is running, its status can be queried, including all startup parameters, running process information and current execution status. This process information consists of hostname, process id, and session id. Due to the restriction syntax, all data can be used for matching in commands.

During execution, process groups can also be killed or signaled, with the commands *kill-process-group* and *signal-process-group*, respectively. Once the process group was exited, a client can call *wait-process-group* to collect exit codes and output. This operation also purges all traces of the process-group from the system.

## 2 Commands

- *create-process-group*: This command starts a new process group. All startup parameters are specified in this command, including executable, user, path, cwd, environment variables, and locational information. This command returns a process group element conforming to the restriction syntax.
- *get-process-group*: This command gets process information, including all specifications passed in to *create-process-group*. This command also returns current job state and an element for each currently executing process. Each currently executing process-group will be returned at most once, depending on the specification.

- *signal-process-group signal='SIGNAL'*: Signals each process-group matching the specification with the signal SIGNAL once.
- *kill-process-group*: Kills each process group matching the specification. Each matching group is killed exactly once.
- *wait-process-group*: Expunge all information about matching process groups from the process manager. Each matching process-group is expunged and returned exactly once.

A usual process group execution consists of a *create-process-group* command followed by polling of *get-process-group* to determine current execution status until the process has exited. Finally, a *wait-process-group* is executed, which can return process output and exit status, and can remove all traces of the process group from the process-manager.

### 3 Events

The process manager emits two events, one upon process-group startup, and once upon completion. In this example, events pertain to the process group with pgid 29. These events look like:

```
<event component='process-manager' msg='process_start' data='29' />
<event component='process-manager' msg='process_end' data='29' />
```

### 4 Examples

In this section, we give examples of a typical process group execution. This process usually consists of a *create-process-group* command, with an event-manager subscription delivering pertinent process-manager events. Once a process\_end event is delivered, the client calls *wait-process-group* to collect exit status and clean up after the process. The initial command takes the form:

```
<create-process-group pgid='*' submitter='desai' totalprocs='8' output='discard'>
  <process-spec exec='/bin/true' cwd='/' range='0-5' />
  <process-spec exec='/bin/false' cwd='/tmp' />
</create-process-group>
```

This command creates a process group consisting of 8 processes. These processes run as user desai, and output is discarded. The specification of pgid causes the process group id, allocated on the process manager, to be returned in the response. The two process-spec elements describe the processes that make up the process group. In this case, 6 processes have the executable */bin/true*, while 2 have the executable */bin/false*. These executables are also run with different current working directories. The following message is sent as a response to the *create-process-group* request.

```

<process-group pgid='29' submitter='desai' totalprocs='8' output='discard'>
    <process-spec exec='/bin/true' cwd='/' range='0-5' />
    <process-spec exec='/bin/false' cwd='/tmp' />
</process-group>

```

This response contains all fields contained in the request, with the pgid returned, since it is allocated in the process-manager component.

During process-group execution, the current status of the process-group can be requested as follows. During execution, the process-group datatype contains all fields specified in the *create-process-group* invocation, with the addition of a status attribute. This attribute is either “running”, corresponding to a currently executing process-group, or “finished”, corresponding to a process-group that has finished execution, but has not yet been reaped with the *wait-process-group* command. The process-group datatype also contains child elements for each process currently executing as a part of the process-group. These process child elements have three attributes, host, corresponding to execution location, pid, corresponding to the process id, and session, corresponding to session id. All characteristics of the process group datatype can be queried in any command, allowing matching against process execution location, and so forth. A query to find locations of process group execution would look like the following:

```

<get-process-group>
    <process-group pgid='29'>
        <process host='*' />
    </process-group>
</get-process-group>
\begin{verbatim}

```

This query requests information about the location of process running as a part of process-group 29. The response looks like:

```

\begin{verbatim}
<process-groups>
    <process-group pgid='29'>
        <process host='ccn1' />
        <process host='ccn2' />
        <process host='ccn4' />
        <process host='ccn2' />
        <process host='ccn5' />
        <process host='ccn7' />
        <process host='ccn9' />
        <process host='ccn12' />
    </process-group>
</process-groups>

```

Clients can either poll for process-group completion or rely on event subscriptions to provide asynchronous notifications of process group completion.

In the polling case, the following command will return the current execution status of process-group 29.

```
<get-process-group>
  <process-group pgid='29' status='*' />
</get-process-group>
```

Once the process group has completed execution, the response will look like:

```
<process-groups>
  <process-group pgid='29' status='finished' />
</process-groups>
```

Another option is to match both fields against their desired state. In this scenario, once the process group has completed, a non-empty response will be returned. This query looks as follows:

```
<get-process-group>
  <process-group pgid='29' status='finished' />
</get-process-group>
```

The response during execution will be:

```
<process-groups/>
```

Once the process group has finished, the same response as above will be returned. Once the process-group has completed, it must be reaped. This command looks like:

```
<wait-process-group>
  <process-group pgid='29'>
    <output/>
    <error/>
    <exit-status rank='*' host='*' pid='*' status='*' />
  </process-group>
</wait-process-group>
```

This command waits on process-group 29, and requests that standard output, error, and exit status information be returned. In this case, we specified that output be discarded, so output and error will both be null in the response. For our process, the response would look like:

```
<process-groups>
  <process-group pgid='29'>
    <output/>
    <error/>
    <exit-status host='ccn1' pid='423' rank='0' status='1' />
    <exit-status host='ccn2' pid='2312' rank='1' status='0' />
```

```

<exit-status host='ccn4' pid='253' rank='2' status='0' />
<exit-status host='ccn2' pid='4763' rank='3' status='1' />
<exit-status host='ccn5' pid='8423' rank='4' status='2' />
<exit-status host='ccn7' pid='2365' rank='5' status='0' />
<exit-status host='ccn9' pid='9324' rank='6' status='0' />
<exit-status host='ccn12' pid='3874' rank='7' status='0' />
</process-group>
</process-groups>
```

This command results in information for all component processes in process-group 29. If instead the client were only interested in processes that exited with non-zero exit status, the following command could be executed instead.

```

<wait-process-group>
  <process-group pgid='29'>
    <exit-status host='eq:*' status='!=0' match='complex' />
  </process-group>
</wait-process-group>
```

This command only returns exit status information for processes that returned a non-zero exit status, and the hostnames that executed these processes. The response would look like:

```

<process-groups>
  <process-group pgid='29'>
    <process host='ccn1' status='1' />
    <process host='ccn2' status='2' />
    <process host='ccn5' status='2' />
  </process-group>
</process-groups>
```

At the conclusion of this process, pgid is expunged from the system, and all traces of the process-group are gone.

Alternatively, process-groups can be signalled, or directly killed during execution without waiting for normal process termination. These commands, *kill-process-group* and *signal-process-group* operate as follows.

```

<kill-process-group>
  <process-group pgid='29' />
</kill-process-group>
```

This command kills process-group 29, and only returns the process group id. Note that the process-group must still be reaped after this command. Similarly, process-groups can be signaled as follows:

```

<signal-process-group signal='SIGKILL'>
  <process-group pgid='29' />
</signal-process-group>
```

The preceding command signals process-group 29 with the signal SIGKILL. Similarly to *kill-process-group* process-group 29 still must be manually reaped.

## 5 Schemas

The schemas for the process manager are split into three parts. One part contains the datatype definitions for all process manager data. Another provides a schema for commands sent to the process manager. The third provides a schema for the responses sent by the process manager to all commands.

### 5.1 Datatypes Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xml:lang="en">
  <xsd:annotation>
    <xsd:documentation>
      Process Manager component schema
      SciDAC SSS project, 2002 Andrew Lusk alusk@mcs.anl.gov
    </xsd:documentation>
  </xsd:annotation>

  <xsd:simpleType name='rlimitType'>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="core|cpu|fsize|data|stack|rss|nproc|nofile|ofile|memlock|as|\*"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name='outputType'>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="merged|discard|single|\*"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name='pmstate'>
    <xsd:restriction base='xsd:string'>
      <xsd:pattern value="running|finished|\*"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name='limitType'>
    <xsd:attribute name='type' type='rlimitType' use='required' />
    <xsd:attribute name='value' type='xsd:string' use='required' />
  </xsd:complexType>

  <xsd:complexType name="argType">
    <xsd:attribute name="idx" type="xsd:string" use="required" />
    <xsd:attribute name="value" type="xsd:string" use="required" />
  </xsd:complexType>

  <xsd:complexType name="envType">
```

```

<xsd:attribute name="name" type="xsd:string" use="required"/>
<xsd:attribute name="value" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="pg-spec">
  <xsd:choice minOccurs='0' maxOccurs='unbounded'>
    <xsd:element name="arg" type="argType"/>
    <xsd:element name="env" type="envType"/>
    <xsd:element name="host-spec" type="xsd:string"/>
  </xsd:choice>
  <xsd:attribute name="range" type="xsd:string"/>
  <xsd:attribute name="user" type="xsd:string"/>
  <xsd:attribute name="co-process" type="xsd:string"/>
  <xsd:attribute name="exec" type="xsd:string" use="required"/>
  <xsd:attribute name="cwd" type="xsd:string" use="required"/>
  <xsd:attribute name="path" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="pg-spec-restriction">
  <xsd:choice minOccurs='0' maxOccurs='unbounded'>
    <xsd:element name="arg" type="argType"/>
    <xsd:element name="env" type="envType"/>
    <xsd:element name='limit' type='limitType' />
    <xsd:element name="host-spec" type="xsd:string"/>
  </xsd:choice>
  <xsd:attribute name="range" type="xsd:string"/>
  <xsd:attribute name="user" type="xsd:string"/>
  <xsd:attribute name="co-process" type="xsd:string"/>
  <xsd:attribute name="exec" type="xsd:string"/>
  <xsd:attribute name="cwd" type="xsd:string"/>
  <xsd:attribute name="path" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="procType">
  <xsd:attribute name="host" type="xsd:string" use="required"/>
  <xsd:attribute name="pid" type="xsd:string" use="required"/>
  <xsd:attribute name="exec" type="xsd:string" use="required"/>
  <xsd:attribute name="session" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="procRestrictionType">
  <xsd:attribute name="host" type="xsd:string"/>
  <xsd:attribute name="pid" type="xsd:string"/>
  <xsd:attribute name="exec" type="xsd:string"/>
  <xsd:attribute name="session" type="xsd:string"/>
</xsd:complexType>

```

```

<xsd:complexType name="ExitType">
  <xsd:choice maxOccurs='unbounded' minOccurs='0'>
    <xsd:element name="exit-code">
      <xsd:complexType>
        <xsd:attribute name='rank' type="xsd:string"/>
        <xsd:attribute name='status' type="xsd:string"/>
        <xsd:attribute name='host' type="xsd:string"/>
        <xsd:attribute name='pid' type="xsd:string"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="pgType">
  <xsd:choice minOccurs="1" maxOccurs="unbounded">
    <xsd:element name="process-spec" type="pg-spec"/>
    <xsd:element name='output' type='xsd:string' />
    <xsd:element name='error' type='xsd:string' />
    <xsd:element name="host-spec" type="xsd:string"/>
    <xsd:element name="exit-status" type="ExitType"/>
  </xsd:choice>
  <xsd:attribute name="pgid" type="xsd:string" use="optional"/>
  <xsd:attribute name="state" type="pmstate" use="optional"/>
  <xsd:attribute name="submitter" type="xsd:string" use="required"/>
  <xsd:attribute name="totalprocs" type="xsd:string" use="required"/>
  <xsd:attribute name="output" type="outputType" use="required"/>
</xsd:complexType>

<xsd:complexType name="pgRestrictionType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="process-spec" type="pg-spec-restriction"/>
    <xsd:element name='output' type='xsd:string' />
    <xsd:element name='error' type='xsd:string' />
    <xsd:element name="host-spec" type="xsd:string"/>
    <xsd:element name="process" type="procRestrictionType"/>
    <xsd:element name="exit-status" type='ExitType' />
  </xsd:choice>
  <xsd:attribute name="pgid" type="xsd:string"/>
  <xsd:attribute name='output' type='outputType' />
  <xsd:attribute name="submitter" type="xsd:string" />
  <xsd:attribute name="totalprocs" type="xsd:string" />
  <xsd:attribute name="state" type="pmstate" />
</xsd:complexType>

</xsd:schema>

```

## 5.2 Inbound Command Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xml:lang="en">
  <xsd:annotation>
    <xsd:documentation>
      Process Manager component inbound schema
      SciDAC SSS project, 2002 Andrew Lusk alusk@mcs.anl.gov
    </xsd:documentation>
  </xsd:annotation>

  <xsd:include schemaLocation="pm-types.xsd"/>

  <xsd:complexType name="createpgType">
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
      <xsd:element name="process-spec" type="pg-spec"/>
      <xsd:element name="host-spec" type="xsd:string"/>
    </xsd:choice>
    <xsd:attribute name="submitter" type="xsd:string" use="required"/>
    <xsd:attribute name="totalprocs" type="xsd:string" use="required"/>
    <xsd:attribute name="output" type="outputType" use="required"/>
    <xsd:attribute name="pgid" type="xsd:string" use="optional"/>
  </xsd:complexType>

  <xsd:element name="create-process-group" type="createpgType"/>

  <xsd:element name="get-process-group-info">
    <xsd:complexType>
      <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="process-group" type="pgRestrictionType"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="signal-process-group">
    <xsd:complexType>
      <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="process-group" type="pgRestrictionType"/>
      </xsd:choice>
      <xsd:attribute name="signal" type="xsd:string" use="required"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="kill-process-group">
    <xsd:complexType>
      <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="process-group" type="pgRestrictionType"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
```

```

        </xsd:choice>
    </xsd:complexType>
</xsd:element>

<xsd:element name="wait-process-group">
    <xsd:complexType>
        <xsd:choice minOccurs="1" maxOccurs="unbounded">
            <xsd:element name="process-group" type="pgRestrictionType"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>

</xsd:schema>

```

### 5.3 Response Schema

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xml:lang="en">
    <xsd:annotation>
        <xsd:documentation>
            Process Manager component outbound schema
            SciDAC SSS project, 2002 Andrew Lusk alusk@mcs.anl.gov
        </xsd:documentation>
    </xsd:annotation>

    <xsd:include schemaLocation="pm-types.xsd"/>
    <xsd:include schemaLocation="sss-error.xsd"/>

    <xsd:element name="process-groups">
        <xsd:complexType>
            <xsd:choice minOccurs='0' maxOccurs='unbounded'>
                <xsd:element name="process-group" type="pgRestrictionType"/>
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="process-group" type="pgRestrictionType"/>

    <xsd:element name="error" type="SSSErrorType"/>

</xsd:schema>

```