S4PM 5.7.1 Installation and Configuration Guide

A guide to installing and configuring NASA's open source Simple, Scalable, Script-Based, Science Processor for Measurements (S4PM)

August 2005

Document Version 1.0.0

Stephen W. Berrick, NASA

Table of Contents

1. Introduction

This document describes the installation and configuration process for S4PM, version 5.7.1.

The Simple, Scalable, Script-based Science Processor for Measurements (S4PM) is a NASA developed system for highly automated processing of science data. S4PM is the main processing engine at the Goddard Earth Sciences Data and Information Services Center (GES DISC). In addition to being scalable up to large processing systems such as the GES DISC, it is also scalable down to small, special-purpose processing strings.

S4PM consists of two main parts: the kernel is the Simple, Scalable, Script-based Science Processor (S4P), an engine, toolkit and graphical monitor for automating script-based, data-driven processing. The S4PM system is built on top of S4P and implements a fully functioning processing system that supports a variety of science processing algorithms and scenarios.

S4PM requires Perl (ideally 5.6 or higher) and has been run successfully on Irix, Linux (RedHat), Solaris, Macintosh OS X, and Microsoft Windows.

S4PM was released to the open source community under the NASA Open Source Agreement in April 2005 with version 5.6.2. The software is available from SourceForge at this URL: http://sourceforge.net/projects/s4pm/.

1.1 Goals of S4PM

The main goal of S4PM is to automate science processing to the extent that a single operator can monitor all of the processing in an "industrial-size" data processing center. A second goal is to be flexible enough to easily add new processing strings or new algorithms to an existing string with a minimum of effort.

High usability is another key goal of S4PM, deriving from the need for more automation at less operational cost. Specific goals are:

- Allow a single operator to manage and monitor hundreds of jobs simultaneously.
- Drill down to troubleshoot a problem in two mouse clicks.
- Set up a new processing string in less than 30 minutes.

1.2 Future Directions

The architecture of S4PM and S4P was specifically designed to be highly modular so that it could evolve quickly and flexibly. It has already evolved from data-driven processing of MODIS instrument data to AIRS processing to on-demand subsetting based on user

requests. Version 5.7.0 was the first release incorporating data mining into S4PM, allowing users to upload algorithms via a Web interface for execution at the GES DISC.

For the future, S4PM will evolve to:

- Support an ever-increasing variety of processing algorithms, scenarios and data interfaces.
- Increase the automation of failure monitoring and recovery.
- Reduce the time and expertise needed to setup and adapt S4PM to new processing algorithms.

We hope that some or all of these goals will be reached by collaborating with the open source community.

2. Related Documentation

The S4PM home page is at: http://disc.gsfc.nasa.gov/techlab/s4pm/ where the following documents are available:

- The S4PM 5.7.1 Operations Guide
- The S4PM 5.7.1 Design Document

3. Installing S4PM

This section describes how to download and install S4PM.

S4PM is available on SourceForge at http://sourceforge.net/projects/s4pm/

There are three files to download:

1. S4PM-5.7.1.tar.gz
2. S4P-5.7.1.tar.gz
3. S4PM_CFG-5.7.1.tar.gz

Download these three files into some directory on the machine where you will install S4PM. The 5.7.1 refers to the most current stable S4PM version. The directory you download these files into is only used for installing S4PM and can be removed later.

Unzip and untar each of the three files. On Linux, you can untar and unzip with one command:

```
tar xvzf S4PM-5.7.1.tar.gz
tar xvzf S4P-5.7.1.tar.gz
tar xvzf S4PM_CFG-5.7.1.tar.gz
```

On other UNIX machines, you may have to unzip and untar separately:

```
gunzip S4PM-5.7.1.tar.gz && tar xvf S4PM-5.7.1.tar
gunzip S4P-5.7.1.tar.gz && tar xvf S4P-5.7.1.tar
gunzip S4PM_CFG-5.7.1.tar.gz && tar xvf S4PM_CFG-5.7.1.tar
```

Unpacking these tar files will result in three subdirectories: S4P-5.7.1, S4PM-5.7.1, and S4PM_CFG-5.7.1.

Change directories into the S4P-5.7.1 directory first:

```
cd S4P-5.7.1
```

For installation of the binaries into the standard system directories on your machine, run the following:

```
perl Makefile.PL
make
make test (optional)
make install
make clean (optional)
```

If you want to install into a non standard directory, instead use:

```
perl Makefile.PL PREFIX=<alternate_directory>
make
```

```
make test
make install
make clean (optional)
```

Change directories into the S4PM_CFG-5.7.1 next and then run the same steps as above.

Finally, change directories into the S4PM-5.7.1 next and then run the same steps as above.

> → **NOTE:** Be aware that if you installed into a non standard directory (using PREFIX above), the S4PM user will have to set the PERLLIB environment variable to the alternate location of the S4PM libraries and the PATH environment variable the location where the binaries get installed (in a shell start-up script, for example).

For example, to install the S4PM binaries and libraries under /home/jdoe rather than in the standard system directories, you will need to follow these steps:

1. Log in as the S4PM user that will run S4PM.

2. Set the PERLLIB environment variable to the location where the libraries are to be installed. For example (in Bourne, Korn, Bash shell or their variants):

```
export PERLLIB=/home/jdoe/lib/perl5/site_perl/5.8.3:
/home/jdoe/lib/perl5/site_perl/5.8.3/i386-linux-
thread-multi
```

3. Run the install:

```
perl Makefile.PL PREFIX=/home/joe
make
make test
make install
make clean (optional)
```

4. Set the PATH environment variable to include /home/joe/bin.

4. Stringmaker Overview

This section describes Stringmaker and how it is used to configure and set up S4PM strings. It is assumed that S4PM has been installed properly as described in Section 3.

Since Stringmaker, as the name implies, is a tool to build and set up S4PM strings, it is prudent to first define what is meant by a string. An S4PM string is a single instance of S4PM, its complement of stations, and the algorithms configured to run in that string. More than one S4PM string can be set up on a single machine. Why have more than one string? A classic reason to separate strings is if you want one string to be for real-time processing driven by event notification (data via subscription) and the other to be for reprocessing or for filling in holes and gaps with data being ordered. In such a case, the algorithms may be the same although the production rules may differ.

Another reason may be that you have algorithms that do not interact with each other (*e.g.* supporting two different missions) or the algorithms logically fall into distinct groups (*e.g.* again, supporting different missions). In this case, it may be appealing to separate them into multiple S4PM strings.

On the other hand, in cases where the same data are used by multiple sets of algorithms, it may pay to have them run within the same string to minimize costly data transfers.

4.1 Why Stringmaker?

Before using Stringmaker, you will need to answer several questions regarding S4PM at your site:

1. Will S4PM be interfacing with ECS at your site? S4PM doesn't require ECS.

2. Will there ultimately be more than one S4PM string configured either on the same machine or on several machines?

3. What type of S4PM string do you want? There are several flavors:

    a. Near real-time processing driven by (nominally) ECS subscriptions.
    b. Reprocessing in which data for a period of time are manually ordered.
    c. On-demand processing in which events (*e.g.* someone making a request through a client) drive production.

S4PM strings are nothing more than a number of directories representing stations with each directory containing one or more scripts and configuration files. It is possible to create a string by hand. This is, however, a laborious task since it involves many manual steps, a mistake in any of which could render the S4PM string useless.

Stringmaker (and its predecessor, Stringmaster) was created to alleviate this burden by automating the process of creating and modifying strings. Stringmaker can handle any flavor of S4PM.

4.2 Preparation For Stringmaker

You will have to make several decisions before configuring and running Stringmaker:

- Who will be the user running S4PM? This same user needs to be the one running Stringmaker. In order for Stringmaker to be responsible for multiple strings, the same user needs to be used for all strings.
- If you will be running S4PM on multiple machines, you will need to have some directory location that is visible across all these machines. Typically, it is the home directory of the S4PM user.
- The location of the algorithms to be run in S4PM will have to be visible to the S4PM user and be granted the correct permissions to be executed by the S4PM user. Any static files used by the algorithms will need to be readable by the S4PM user. Algorithm locations can be different for each S4PM string.
- For S4PM strings that need to interface with ECS:
    - If you wish to configure a string to get data from the ECS via requests for those data, S4PM uses the ECS Science Data Server Command-Line Interface (SCLI). Distribution Notifications (DNs) are sent via e-mail to the S4PM user once the data have been pushed. In order for these DNs to be processed, the S4PM user needs to direct e-mailed DNs to the Receive DN station of the string for whom the data were ordered. This is best accomplished with a procmail filter.
    - For subscription based processing with data from the ECS, the S4PM user will need to subscribe to ECS notifications of insert of needed data types. These notifications are e-mailed to the S4PM user and, as above, procmail is the most efficient way to direct those e-mails to the Subscription Notify station. Note that this station sits across all S4PM strings.

4.3 Stringmaker Configuration Files

Stringmaker is a Perl script that builds S4PM strings based on configuration parameters set in several hierarchical configuration files. Most of the work in getting Stringmaker to build the strings you want are in setting up these configuration files. Once set up properly, S4PM strings can be created or modified easily.

The Stringmaker configuration files are described below. Stringmaker reads these configuration files in the order shown. The configuration files are organized so that the most global parameters are specified at the top of the configuration file chain and whereas the more specific ones are specified at the bottom.

4.3.1 The Stringmaker Global Configuration File

The global configuration file is named s4pm_stringmaker_global.cfg and contains parameters that are common across all S4PM strings. Anything in this file, however, can be overridden in any of the following configuration files.

Section 5 has a detailed discussion on the Stringmaker global configuration file.

4.3.2 The Stringmaker Host Configuration File

The host configuration file contains parameters that are common to a particular host machine, but that may differ from one machine to another. The actual file name for this configuration file is the host machine name with the .cfg file name extension. The host machine name is the same as what the 'uname -n' UNIX command would return. For example, g0spg11.cfg. There needs to be one such configuration file for each machine on which S4PM is to be installed.

Anything in this file can be overridden in any of the following configuration files.

Section 6 has a detailed discussion on the Stringmaker host configuration file.

4.3.3 The Stringmaker Data Types Configuration File

The data types configuration file is named s4pm_stringmaker_datatypes.cfg and contains data type parameters for all S4PM strings. It is intended to be a pool from which individual strings draw information about data types.

As above, anything in this file can be overridden in any of the following configuration files.

Section 7 has a detailed discussion on the Stringmaker data types configuration file.

4.3.4 The Stringmaker Static Configuration File

The static configuration file is named s4pm_stringmaker_static.cfg and, unlike the ones above, is not meant to be modified. It is intended to be static as its name implies. It is in this file where a number of the S4PM stations are described and their configuration files set.

Section 8 has a detailed discussion on the Stringmaker static configuration file.

4.3.5 The Stringmaker String Configuration File

This configuration file is unique for each individual S4PM string and is meant to specify parameters unique to a string. Unlike with the above configuration files, the file name is

completely arbitrary although a consistent naming convention is recommended if your site has multiple strings.

Among other things, this configuration file sets what algorithms are to be run. It is assumed that algorithms listed here have their own algorithm configuration files (see Section 4.3.6).

Section 9 has a detailed discussion on the Stringmaker string configuration file.

4.3.6 The Stringmaker Algorithm Configuration Files

The algorithm configuration files specify information about the algorithms. There needs to be one such file for each algorithm. The name of file must be the algorithm name followed by an underscore followed by the profile and then the .cfg file name extension.

For example:

MoPGE01_RPROC.cfg

As one would guess, these algorithm configuration files contain parameters having to do with a particular algorithm to be run in S4PM. This includes specifying the data types to be input and output by the algorithm. These data types must exist in the s4pm_stringmaker_datatypes.cfg file. Unlike with all the other Stringmaker configuration files, the algorithm configuration files are part of the algorithm package and reside where the rest of the algorithm package resides.

Section 10 has a detailed discussion on the Stringmaker algorithm configuration files.

4.3.7 The Stringmaker Jobs Configuration File

The jobs configuration file is named s4pm_stringmaker_jobs.cfg and it contains only one parameter. That is, the maximum number of jobs that can be run in a S4PM station in a particular string. Unless set in this file, the maximum number of jobs that can be run in any station is five. For stations where the number needs to be different (higher or lower), this file is used. Note that unlike the other Stringmaker configuration files, this one is optional.

Section 11 has a detailed discussion on the Stringmaker jobs configuration file.

4.3.8. The Stringmaker Derived Configuration File

The derived configuration file is named s4pm_stringmaker_derived.cfg and this configuration file is at the bottom of the hierarchy. Based on all of the above configuration files, this file makes decisions on which stations are to be configured in a particular string and how they are supposed to interact with one another. Like the s4pm_stringmaker_static.cfg file, this file is not meant to be modified.

Section 12 has a detailed discussion on the Stringmaker derived configuration file.

4.3.9 Configuration File Summary

The minimum set of configuration files needed for the simplest S4PM string running a single algorithm is seven:

1. s4pm_stringmaker_global.cfg
2. <host>.cfg
3. s4pm_stringmaker_datatypes.cfg
4. one <algorithm>_<profile>.cfg file
5. s4pm_stringmaker_static.cfg
6. <string>.cfg
7. s4pm_stringmaker_derived.cfg

Of these, you only need to create/modify five of them:

1. s4pm_stringmaker_global.cfg
2. <host>.cfg
3. s4pm_stringmaker_datatypes.cfg
4. one <algorithm>_<profile>.cfg
5. <string>.cfg

The Section 5 through 12 will delve into each of the configuration files in detail.

4.3.10 Running Stringmaker

This section describes how to actually run Stringmaker.

4.3.10.1 Before Running Stringmaker

Before running Stringmaker on an existing string, you need to consider whether or not the string needs quiesced. By quiesced, we mean a state in which all stations in the string are turned off (show up as red in the S4PM Monitor) and there are no jobs running in any of the stations (all jobs are either blue for queued up or red for failed).

To play it safe, always quiesce your string before you run Stringmaker. For small configuration changes, you may get away with not having to do so. This is, however, not recommended.

For particularly large or deep changes, you may even want to run the string “dry” prior to running Stringmaker. That is, allow the string to finish up processing and exporting data from any current and queued jobs, but not allow more data to come into the string.

4.3.10.2 The Stringmaker Command

The commands to run Stringmaker is:

```
s4pm_stringmaker.pl –c|-u|-a  –s <string>.cfg
```

where <string>.cfg is the name of the Stringmaker string configuration file.

With the –c option, a new S4PM string is created. If one already exists, it will be overwritten.

With the –u option instead, all station and script configuration files are created overwriting any that may already exist. With this option, as opposed to the –c, no new links or directories will be created. Thus, you don't want to use the –u option if adding a new or updated algorithm to S4PM (since this involves the creation of some new directories and links).

The –a option is for a very special case. It is only used when changes have been made to the Stringmaker jobs configuration file such as increasing the maximum number of jobs in Run Algorithm in some string. In this case, Stringmaker will only alter the station configuration files of the stations affected by the change and nothing else.

If in doubt, the –c option is always safe and there is almost no performance penalty for running it over the other options.

For example:

```
s4pm_stringmaker.pl –c –s S4PM10_MO_RE.cfg
```

will create (or re-create) a string whose string configuration file is named: S4PM10_MO_RE.cfg.

4.3.11 Using The S4PM Monitor To Install An Algorithm

Stringmaker can be run on the command line as discussed in Section 4.3.10.2 to add (or remove) an algorithm from S4PM. It would simply involve editing the string configuration file to first remove the algorithm from the list in the @run_sorted_algorithms parameter array and then running Stringmaker with the –c option. If only the version of the algorithm changed, then the string configuration file would be edited to change the algorithms version in the %algorithm_versions hash parameter in the same file before running Stringmaker.

But there is an easier way:

From the S4PM Monitor, right-click on the Configurator station button and select Install Algorithm or Uninstall Algorithm. For installation, you will be asked to first select the string you wish to alter and then select the algorithm configuration file corresponding to the algorithm you wish to install. Remember, it is assumed that you already placed the algorithm package unpacked into the correct location.

Uninstalling works much the same way. You will be shown a list of algorithms currently configured for this string. Select the one you wish to delete. Note that the algorithm package will *not* be deleted from disk; only S4PM will be configured not to run it.

Underneath the covers, it is Stringmaker that gets run with installing or uninstalling an algorithm in this manner. The advantage is that the string does not need to be quiesced or run dry. The appropriate stations will reconfigure themselves for the new algorithm (or lack thereof).

→ **NOTE:** One caveat you should be aware. If you uninstall an algorithm for which jobs corresponding to that algorithm are still being processed in the string, those jobs will ultimately fail since they will be passed to a station that has, in the interim, lost all memory of that algorithm. This isn't a problem, but you may opt to first let jobs corresponding to the algorithm work themselves out before initiating the uninstall.

5. The Stringmaker Global Configuration File

The Stringmaker global configuration file is meant for parameters that are global across all S4PM strings at a particular site. For sites that will install S4PM on multiple host machines, some consideration needs to be given for how production will be parceled to these strings.

5.1 File Name

The file name for the Stringmaker global configuration file is:

s4pm_stringmaker_global.cfg

5.2 $user

This parameter is MANDATORY.

The $user parameter is the name of the user account that will be managing and running all S4PM strings. This S4PM user will own all files in the string. Stringmaker itself needs to be run as this user.

Example:

```
$user = 's4pmuser';
```

5.3 $s4pm_email

This parameter is OPTIONAL, but required if $input_symlink_root is set in the Stringmaker String configuration file (see Section 9).

The $s4pm_email is the e-mail address of the S4PM user $user. Remember to escape the @ symbol when setting it as in the following example:

```
$s4pm_email = 's4pmuser\@myhome.com';
```

5.4 $global_root

This parameter is MANDATORY.

The $global_root parameter is a root directory that is visible across all S4PM strings at sites that support multiple S4PM strings running on multiple machines. Typically, this variable is set to a cross-mounted directory such as the home directory of the S4PM user or some directory therein. For sites where S4PM strings reside only on a single machine,

this variable can be set to $s4pm_root (see Section 4.4.1.3). The default is $HOME directory of the user running Stringmaker.

Example:

```
$global_root = "/home/s4pmuser/s4pm";
```

5.5 $stringmaker_root

This parameter is MANDATORY.

When a S4PM string is first created, Stringmaker needs to do so from within a designated directory where the Stringmaker configuration files reside (with the exception of the algorithm configuration files which reside with the algorithms). This designated directory then becomes the directory of the Configuration station in the string created. The Configurator station can be viewed as the manifestation of Stringmaker within an S4PM string.

The parameter $stringmaker_root is the directory from which Stringmaker is run and is also the station directory of the Configuration station. As such, this directory must be visible across all machines that host strings that are to be managed by Stringmaker and Configurator (such as the cross-mounted home directory of the user running the strings). The default is $HOME/stringmaker of the user running Stringmaker.

Example:

```
$stringmaker_root = "/home/s4pmuser/s4pm/stringmaker";
```

5.6 %run_env_variables

This parameter is OPTIONAL.

The %run_env_variables parameter is an optional hash that allows environment variables to be set for algorithms running in the Run Algorithm stations of all strings. Hash keys are the environment variable names and hash values are their values. Environment variables defined here will apply to all algorithms running in all strings. To have distinct environment variables for each machine, place the %run_env_variables hash in the <host>.cfg file instead. To have distinct environment variables for each string, place the hash in the <string>.cfg file instead. Note that the environment variable PATH is predefined by S4PM and should not be set in this hash.

Example:

```
%run_env_variables = (
'LM_LICENSE_FILE' =>
"/usr/ecs/$mode/COTS/IMSLv3v4/license/license.dat",
'HDFLOOKPATH' => "/tools/gdaac/$mode/bin",
);
```

5.7 $dataserver_ur

This parameter is MANATORY if interfacing with ECS, OPTIONAL otherwise.

The $dataserver_ur parameter is the Universal Reference (UR) of the ECS Science Data Server. This parameter is only needed if S4PM is interfacing with ECS.

Example:

```
$dataserver_ur =
'UR:10:DsShESDTUR:UR:15:DsShSciServerUR:13:[GSF:DSSDSRV]';
```

5.8. @privileged_users

This parameter is OPTIONAL.

The @privileged_users parameter is an array of users that are to be given permissions to execute certain critical functions via the S4PM Monitor. The assumption here is that S4PM is being run under a common user account (e.g. s4pmuser) yet you do *not* want just anyone logged in as 's4pmuser' to execute some very critical functions. The critical functions are shown below:

Critical Function Name	Description
Kill All	Kills all stations (stops them) and kills any jobs running within those stations.
Bypass QA	Force data to be registered within S4PM (Register Data station) even if it fails quality assurance (QA) checking.
Release Job Now	Release a job that is running in the Select Data station while accumulating input data for an algorithm run.
Ignore Optional	Instruct a job to stop looking for any more optional input data for an algorithm (in the Find Data station).
Ignore Required	Instruct a job to stop looking for any more required input data for an algorithm (in the Find Data station).
Expire Current Timer	Instruct a job to give up on the current optional input it is looking for and move on to the next (in the Find Data station).

Table 5-1. Critical operational functions for which the @privileged_users parameter applies.

Users listed in the @privileged_users array will need to supply their own user logon ID and password via a pop-up box. They will need to do this in additional to being logged in as the S4PM user. If the user is in the @privileged_users array *and* the password is correct will the user be allowed to run the task. Otherwise, the user will be denied from running the task.

The user 'any' is reserved to mean any user. This might be useful if you want to use the pop-up box as a sort of confirmation that the task is to be carried out (i.e. are you sure?).

If this array is unset or empty, then no pop-up box will be issued prior to running any of the above tasks. This is the default.

> → **NOTE:** Application of the @privileged_users parameter does not constitute a security measure. It only helps to prevent inadvertent or accidental tasks from being run.

Example:

```
@privileged_users = ('jdoe', 'rjones', 'msmith');
```

5.9 Other Parameters

As alluded to earlier, other parameters discussed later that you find to be common across all S4PM strings can be migrated "up the chain" into the s4pm_stringmaker_global.cfg file.

6. The Stringmaker Host Configuration File

The Stringmaker host configuration file is meant to handle parameters that may be different from one host to another yet are common to all strings on a host. If certain parameters described below are, in fact, global at your site, you can opt to specify them in the s4pm_stringmaker_global.cfg file instead. In the current release, however, the host configuration file is mandatory (although it can be effectively empty).

6.1 File Name

The file name for the Stringmaker host configuration file is:

<host_name>.cfg

where <host_name> is the name of the machine. On UNIX machines, it is equivalent to the output from the ‘uname –n’ command.

6.2 $domain

This parameter is MANDATORY if interfacing with ECS, OPTIONAL otherwise.

The $domain parameter is the Internet domain of the machines within the installation. This assumes that all S4PM strings will be on machines within the indicated network.

Example:

```
$domain = 'gsfcb.ecs.nasa.gov';
```

6.3 $bindir

This parameter is MANDATORY.

The $bindir parameter is the directory where S4PM executables are located. This should have been something you specified when installing S4PM. The location needs to be visible across all S4PM strings on all machines.

Example:

```
$mode = "TS2";
$bindir = "/tools/gdaac/$mode/bin";
```

6.4 $cfgdir

This parameter is MANDATORY.

The $cfgdir parameter is the directory where baselined configuration files are located. As with the $bindir (Section 6.3), this should have been something you specified when installing S4PM. The location needs to be visible across all S4PM strings on all machines.

Note that $cfgdir does not refer to where the Stringmaker configuration files reside. Rather, it is the location where other S4PM configuration files and configuration templates reside after they are installed.

Example:

```
$mode = "TS2";
$cfgdir = "/tools/gdaac/$mode/cfg";
```

6.5 $s4pm_root

This parameter is MANDATORY.

The $s4pm_root parameter is the root directory under which S4PM strings are located. For each string installed on this host, Stringmaker will make unique subdirectories for each string and each instance of a string under this root directory.

Example:

```
$mode = "TS2";
$s4pm_root = "/vol1/$mode/s4pm";
```

6.6 $ingest_root

This parameter is MANDATORY.

The $ingest_root parameter is the root under which PDRs and PANs are exchanged with the ECS. For each string installed on this host, Stringmaker will make unique subdirectories for each string and each instance of a string under this root directory.

Example:

```
$mode = "TS2";
$ingest_root = "/vol1/$mode/s4ins";
```

6.7 $data_root

This parameter is OPTIONAL.

The $data_root parameter is the root under which all data being managed by S4PM reside. This includes data brought in from external sources (e.g. ECS) and data produced within S4PM prior to being exported or distributed. Below $data_root, Stringmaker will make subdirectories for each S4PM string and instance. The default is to put the data root in a directory under $ecs_root (see next) named DATA.

Example:

```
$data_root = '/vol3/data/s4pm';
```

6.8 $ecs_root

This parameter is MANDATORY.

For ECS integration, the $ecs_root parameter is the root directory where the ECS custom code is installed, in particular, the ECS Toolkit, SCLI, and DCLI. This setting also defines where the data root directory is if you don't specify $data_root (see above).

Example:

```
$mode = "TS2";
$ecs_root = "/usr/ecs/$mode/CUSTOM";
```

7. The Stringmaker Data Types Configuration File

The Stringmaker data types configuration file contains information about all data types used in all strings. When a data type is specified in an algorithm configuration file (see Section 10), either as an input or output, Stringmaker assumes that information about this data type is specified in this file. An error is produced if a referenced data type is not in this file.

7.1 File Name

The file name for the Stringmaker data types configuration file is:

s4pm_stringmaker_datatypes.cfg

7.2 %all_datatype_max_sizes

This parameter is MANDATORY.

The %all_datatype_max_sizes parameter is a hash containing maximum sizes in bytes of the corresponding data types listed as the hash keys. For files whose sizes may be highly variable, choose a reasonable maximum. It may be convenient to set up separate hashes first (e.g. one for each mission or S4PM string) and then combine them into the %all_datatype_max_sizes at the end.

For example:

```
%modis_max_sizes = (
   'MOD000' => 352_000_000,
   'MOD01' => 575_000_000,
   'MOD03' => 63_000_000,
);
%airs_max_sizes = (
   'AIRABQAP' => 2_000_000,
   'PMCO_HK' => 2_000_000,
   'PREPQCH' => 75_000_000,
);
%all_datatype_max_sizes = (%modis_max_sizes, %airs_max_sizes);
```

7.3 %all_datatype_versions

This parameter is MANDATORY.

For each data type listed in the %all_datatype_max_sizes hash (Section 7.2), the %all_datatype_versions parameter hash lists data type versions. There must be an entry in this hash for every data type listed in the %all_datatype_max_sizes hash. Furthermore,

data type versions specified in algorithm configuration files must match those set in this configuration file.

Example:

```
map { $all_datatype_versions{$_} = '001'} keys
%all_datatype_max_sizes;
foreach my $dt ( keys %all_datatype_max_sizes ) {
     if ( $dt =~ /^MOD/ and $dt ne 'MOD000' ) {
         $all_datatype_versions{$dt} = '005';
     } elsif ( $dt =~ /^MYD/ and $dt ne 'MODPML0' ) {
        $all_datatype_versions{$dt} = '004';
     } elsif ( $dt =~ /^AI/ ) {
         $all_datatype_versions{$dt} = '002';
     }
};
```

7.4 %ragged_file_trap

This parameter is OPTIONAL.

The %ragged_file_trap parameter is a hash listing those data types that should be trapped if the temporal metadata do not align on the hour boundary. Generally, these are Level 0 data. When such data types are brought into S4PM, they will fail in the Register Data station. Failure handlers are provided to either bypass the trap or have the offending data purged. Hash values must be set to non-zero to enable the trap or zero to disable the trap. Data types not listed at all in this hash are equivalent to setting their values to zero. All data types listed in this hash must appear in the %all_datatype_max_sizes hash.

Example:

```
%ragged_file_trap = map {($_, 1)} (
     'MOD000',
     'MODPML0',
     'AM1ANC',
);
```

7.5 %register_data_offsets

This parameter is OPTIONAL.

The %register_data_offsets parameter is a hash that lists temporal offsets to be applied to data arriving in the Register Data station. The default is to apply no offset.

By default, S4PM names files coming in through Register Data according to the start time as indicated in the accompanying metadata file. If the metadata only lists a single date/time, S4PM uses this value in the file name. Sometimes, however, it is useful to apply an offset to the time as indicated in the S4PM file name, for example to facilitate easier production rules (particularly with model data). This is the primary use for applying offsets.

Example:

```
%register_data_offsets = (
     'OZ_DAILY' => [-12 * 3600, +12 * 3600],
     'SEA_ICE' => [-12 * 3600, +12 * 3600],
);
```

7.6 @all_qc_datatypes

This parameter is OPTIONAL, but MANDATORY is %qc_output (Section 7.7) is set.

The @all_qc_datatypes parameter is an array that lists all data types where quality control (QC) checking should be done. The particular QC checks done are set in the %qc_output hash (next). For simplicity, one may set this array to all data types defined in this file via:

```
@all_qc_datatypes = keys %all_datatype_max_sizes;
```

→ **NOTE:** The $has_qc parameter in the Stringmaker string configuration file (see Table 9-1) controls whether or not QC checking is turned on, regardless of what is in the @all_qc_datatypes array.

7.7 %qc_output

This parameter is OPTIONAL.

The %qc_output parameter is a hash describing the types of QC checking to be performed on data types produced in S4PM (those specified in the @all_qc_datatypes array above; see Section 7.6). Standard QC checking includes s4pm_is_hdf.pl which verifies that an HDF output can be opened as an HDF file (for HDF files only) and s4pm_checksum.pl that computes a checksum for each output and includes that checksum in the output PDR (if $use_checksums is enabled in the <string>.cfg file). Other QC checks may be added. For example, checking file sizes for valid ranges.

The hash keys are data types and the hash values are lists consisting of one or more items in the form:

<bbbbb> <script_command>

where <bbbbb> are 5 one-bit settings that have the following meaning:

- Bit 1 - Apply QC check to metadata file
- Bit 2 - Apply QC check to data file
- Bit 3 - Block export if QC fails

- Bit 4 - Block from Register Local Data if QC fails
- Bit 5 - Fatal (fail the algorithm)

and <script> is the QC script or command to run.

An illustration is:

```
%qc_output = (
   'MOD021KM' => [
     '11111 /tools/gdaac/TS2/bin/s4pm_checksum.pl'
     '11110 /tools/gdaac/TS2/bin/s4pm_check_size.pl -f
../SizesModis.cfg',
   ],
   'MOD01' => [
     '11110 /tools/gdaac/TS2/bin/s4pm_checksum.pl'
     '11110 /tools/gdaac/TS2/bin/s4pm_check_size.pl -f
../SizesModis.cfg',
   ],
   'AM1EPHN0' => [
     '11111 /tools/gdaac/TS2/bin/s4pm_checksum.pl'
   ],
   'MOD35_L2' => [
     '11111 /tools/gdaac/TS2/bin/s4pm_checksum.pl',
   ],
);
```

In the above example, for MOD021KM above, the QC checks are applied to both the data file and the associated metadata files (bits 1 and 2). If a MOD021KM file fails the data size checking, it is blocked from export (bit 3), blocked from going to the Register Local Data station (this effectively blocks it from any upstream processing), bit 4, but because bit 5 is set to zero, the algorithm will not fail in Run Algorithm, although a message will be written to the log file.

7.8 %non_hdf_datatypes

This parameter is OPTIONAL, but MANDTORY is enabling QC on non-HDF data files.

The %non_hdf_datatypes parameter hash is used for marking data types as non-HDF. By default, S4PM assumes that **all** data types are in HDF format. HDF validation is skipped if the data type is listed in this hash. Hash values must be set to non-zero for data types that are non-HDF and to zero (or not set) if the data types are HDF.

Example:

```
%non_hdf_datatypes = map {($_, 1)} (
   'MOD02SSN',
   'MYD02SSN',
);
```

7.9 %skip_checksum_datatypes

This parameter is OPTIONAL.

If check summing is turned on in a particular string (via $use_checksums in the Stringmaker string configuration file), this hash lists data types where check summing should not be done. Hash keys are the data types to skip and hash values should simply be set to non-zero to skip check summing or to zero (or not set) to not skip check summing.

Example:

```
%skip_checksum_datatypes = map {($_, 1)} (
   'MOD35_QC',
   'MYD35_QC',
   'MOD07_QC',
   'MYD07_QC',
   'MYD021QA',
);
```

7.10 %data_file_qa

This parameter is OPTIONAL.

The %data_file_qa parameter is a hash that maps data types to commands to run on those data files to assess quality in the Register Data station. Commands to run are arbitrary and can include scripts, but they must return 0 if the data file passes QA and non-zero otherwise. This QA is run in the Register Data station. A data file that fails QA causes the job in Register Data to fail. The 'Bypass QA' failure handler allows a QA failure to be bypassed; 'Purge Bad-QA Data' allows the offending data to be purged. Other failure handlers for particular QA failures can easily be added.

The distinction between QA here and QC discussed in Section 7.7 is that QA is performed on files coming into the Register Data station while QC is performed on files produced in the Run Algorithm station.

Example:

```
%data_file_qa = (
   'AM1ATTN0' => 's4pm_attitude_check.pl -t .0002',
   'NISE' => 's4pm_nise_check.pl',
);
```

7.11 $s4pm_filename_pattern

This parameter is OPTIONAL.

The $s4pm_filename_pattern parameter is the pattern used by S4PM for constructing file names used internally by S4PM. The pattern is a string containing format specifiers describing how a file name in S4PM is to be built from the data type name and version, the data time, and the production date and time. The format specifiers are based on those used by the UNIX ‘date’ command format option.

If using this optional parameter, the environment variable S4PM_CONFIGDIR ***must*** be set to the location of the Stringmaker configuration files. This is the same as the setting of $stringmaker_root in the Stringmaker global configuration file. If S4PM_CONFIGDIR is not set or if $s4pm_filename_pattern is not set, the file name pattern assumed is the standard S4PM file name pattern.

Format specifiers come in two types, those that begin with the ^ character and those that begin with the ~ character. Format specifiers that begin with the ^ character refer to data time. Format specifiers that begin with the ~ character refer to the current time (same as would be returned via the 'date' command on the machine in which this is running).

Format Specifiers	Description
^E	Data type name
^V	Data type version
^y or ~y	Two-digit year
^Y or ~Y	Four-digit year
^d or ~d	Day of month (00-31)
^m or ~m	Decimal month (00-12)
^b or ~b	Abbreviated month name (Jan, Feb, etc.)
^B or ~B	Long month name (January, February, etc.)
^j or ~j	Day of year (000-366)
^H or ~H	Hours on 24-hour clock (00-23)
^M or ~M	Minutes (00-59)
^S or ~S	Seconds (00-59)
^u or ~u	Decimal day of week (1-7) with 1=Monday
~N	Current time in form: YYYYjjjHHMMSS, a shorthand for ~Y~j~H~M~S. Useful for making file names unique.

Table 7-1. Allowable format specifiers for forming a file name pattern with the $s4pm_filename_pattern parameter. The specifiers starting with the ^ character refer to the data time; those with the ~ character refer to the production (or machine) time.

In addition to format specifiers, the pattern may contain other characters, words or some punctuation (-, :, _). These become fixed in the file names built by S4PM.

→ **NOTE:** When using the $s4pm_filename_pattern parameter, you **MUST** also set the environment variable S4PM_CONFIGDIR to the location of the Stringmaker directory which **MUST** be the same as the setting for $stringmaker_root in the Stringmaker global configuration file. If S4PM_CONFIGDIR is not set or set to a non-existent directory, S4PM will revert to assuming the standard S4PM file name pattern.

7.11.1 File Name Pattern Restrictions

There are certain restrictions when setting up your own file name pattern in S4PM;

1. All file name patterns must contain the data type name and data type version.
2. All file name patterns must make use of the production date and time specifiers (those beginning with ~) to ensure uniqueness of file names.
3. The data date must be included in the file name pattern (even if the time is not).

The default value for $s4pm_filename_pattern is:

```
$s4pm_filename_pattern = "^E.A^Y^j.^H^M.^V.~N.hdf";
```

which produces the standard S4PM file name, for example:

MOD01.A2005067.0340.005.2005167124454.hdf

Here's another example:

```
$s4pm_filename_pattern = "^H^M-^Y^j.~H~M~S.^E.^V.~Y~j.dat";
```

which results in file names like:

0735-2000270.142731.MOD01.005.2005167.dat

8. The Stringmaker Static Configuration File

The Stringmaker static configuration file should not need to be modified (as its name implies). This section will, however, discuss some of the details of this configuration file in case you do find a reason to modify it.

8.1 File Name

The file name for the Stringmaker static configuration file is:

s4pm_stringmaker_static.cfg

The Stringmaker string configuration file specifies how to set up some of the S4PM stations. Only those stations that exist for any S4PM configuration are specified in this file. In some cases, some aspects of a station may be specified here whereas the rest is specified in the s4pm_stringmaker_derived.cfg file (discussed later). Since this configuration file is read in before any of the Stringmaker string configuration file is read, anything having to do with particular data types or algorithms are not known to Stringmaker at this time. Therefore, only those stations that can be set up without this information (at least in part) are set up here.

The Stringmaker string configuration file is broken up into sections for each station that gets specified. Within each station section, many aspects of the station are described. The contents of the station station.cfg files are set in a very intuitive manner that can seen below. In addition, mechanisms for specifying the symbolic links that need to exist in each station as well as other aspects are shown as well.

The list of mechanisms shown below is not exhaustive, but only represent a sampling of the most commonly used ones. In addition, the same mechanisms described below for setting up stations are used in the Stringmaker derived configuration file. Some of the examples, in fact, were taken from the Stringmaker derived configuration file.

8.2 %stations

Most of the information defined in the Stringmaker static (and derived) configuration files is contained in the %stations hash. This hash contains a number of attributes that define particular aspects of each station. Attribute names are either literals or names of Perl variables. In either case, the way in which attribute *X* is set to value *Y* for station *Station* is as follows:

```
$stations{'Station'}{'X'}
```

In the sections below, the various attributes of the %stations hash are discussed.

8.2.1 $cfg_station_name

The $cfg_station_name attribute of the %stations hash defines the station name for the station. Stringmaker will use this value for the $cfg_station_name parameter in the station.cfg file that it builds for this station.

Example:

```
$stations{'register_data'}{'$cfg_station_name'} = 'Register Data';
```

8.2.2 $cfg_disable

The $cfg_disable parameter defines the value of the $cfg_disable parameter in the station.cfg file for the particular station. If $cfg_disable is set to non-zero, Stationmaster will consider the station disabled and non-participating in the string. If set to 0 or unset, Stationmaster will consider the station enabled.

Example:

```
$stations{'register_local_data'}{'$cfg_disable'} = 0;
```

8.2.3 exec_symlinks

The exec_symlinks attribute is set to a list of executables that need to exist as symbolic links in the station. Symbolic links are linked to the location where the S4PM binaries have been installed (this directory is set by the $bindir parameter in the Stringmaker global configuration file; see Section 5).

Example:

```
$stations{'prepare_run'}{'exec_symlinks'} =
['s4pm_prepare_run.pl',
's4pm_prepare_run_resync.pl'];
```

8.2.4 misc_symlinks

The misc_symlinks attribute is set to a hash of miscellaneous symbolic links that need to exist in the station (other than those for executables covered by the exec_symlinks attribute). Unlike with executable symbolic links (Section 8.2.3), the link as well as what it is linking too need to be specified.

Example:

```
$stations{'repeat_daily'}{'misc_symlinks'} = {
   's4pm_allocate_disk.db' =>
'../allocate_disk/s4pm_allocate_disk.db',
   's4pm_allocate_disk.cfg' =>
```

```
'../allocate_disk/s4pm_allocate_disk.cfg'
};
```

8.2.5 $cfg_max_children

The $cfg_max_children parameter sets the maximum number of jobs (children) that can be run in the station at a time. The default, if not set here, is 5.

For example:

```
$stations{'repeat_daily'}{'$cfg_max_children'} = 8;
```

8.2.6 %cfg_commands

The %cfg_commands parameter is a hash that specifies the commands that are to be run in a station with the associated work order types. It is what the %cfg_commands parameter is set to in the station.cfg files.

For example:

```
$stations{'sweep_data'}{'%cfg_commands'} = {
   'CLEAN' => '../s4pm_sweep_data.pl -config
../s4pm_allocate_disk.cfg -db ../s4pm_allocate_disk.db',
};
```

8.2.7 %cfg_downstream

The %cfg_downstream parameter is a hash that sets the stations to which output work orders are directed. It is what the %cfg_downstream parameter is set to in the station.cfg files.

Example:

```
$stations{'repeat_hourly'}{'%cfg_downstream'} = {
   'REPEAT_CLEAN_FILES' => ['repeat_hourly'],
   'ROLLUP_RUSAGE' => ['repeat_hourly'],
   'UPDATE' => ['track_data'],
};
```

8.2.8 %cfg_interfaces

The %cfg_interfaces parameter is a hash that sets the %cfg_interfaces parameter in the station.cfg file. The %cfg_interfaces hash maps button names (which appear in the S4PM Station Monitor window for a particular station or by right-clicking on the station name in the S4PM Monitor) to actions to be carried out. Typically, these are used for bringing up additional window applications (hence the name), but this is not required. The "thing" run can be any command.

For example:

```
$stations{'sweep_data'}{'%cfg_interfaces'} = {
   'Restart All Failed Jobs' => 's4p_restart_all_jobs.pl',
};
```

8.2.9 %cfg_failure_handlers

The %cfg_failure_handlers parameter is a hash that sets the %cfg_failure_handlers parameter in the station.cfg file. The %cfg_failure_handlers maps failure handler names to scripts or commands to run when invoked. Such failure handlers are only available via the S4PM Job Monitor window when a job fails (access it by clicking on the red failed job box).

For example:

```
$stations{'receive_dn'}{'%cfg_failure_handlers'} = {
   'Remove Job' => 'remove_job.pl',
};
```

8.2.10 %cfg_manual_overrides

The %cfg_manual_overrides parameter is a hash that sets the %cfg_manual_overrides parameter in the station.cfg file. The %cfg_manual_overrides maps button names to tasks that carried out in a running job directory. The tasks can be scripts or commands.

For example:

```
$stations{'select_data'}{'%cfg_manual_overrides'} = {
   'Release Job Now' => 'touch RELEASE_JOB_NOW',
   'Modify Timer' => 'touch MODIFY_TIMER',
   'Modify Threshold' => 'touch MODIFY_THRESHOLD',
};
```

9. The Stringmaker String Configuration File

The Stringmaker string configuration file is unique for each S4PM string. The string configuration file is the configuration file where the algorithms to run, along with their versions, and profiles are set. It is also where disk pools are sized. It is based upon the algorithms selected in this configuration file that Stringmaker knows what algorithm-specific configuration files to later read in.

9.1 File Name

There is no requirement for the actual file name although the recommendation is to name the file for the $string_id parameter contained therein (see Section 9.2).

9.2 $string_id

This parameter is MANDATORY.

The $string_id parameter is an identifier for the string, used in both the Ingest polling configuration and the USERSTRING for data requests. This is also the work order pattern for work order in the Receive PAN station.

Example:

```
$stringid = "S4PM10_MO_FW";
```

9.3 $data_source

This parameter is MANDATORY.

The $data_source parameter is used to name the subdirectory under $s4pm_root (set in the host or global configuration file) for this string. Thus, it serves as another identifier for the string.

Example:

```
$data_source = 'terra';
```

9.4 $data_source_longname

This parameter is OPTIONAL.

The $data_source_longname parameter is a longer version of $data_source, a string describing the data source corresponding to $data_source (Section 9.3). It is used in the S4PM Monitor window title bar. If not specified, it is set to $data_source.

Example:

```
$data_source_longname = "MODIS Terra";
```

9.5 $instance

This parameter is MANDATORY.

The $instance parameter represents a subdivision under $data_source. Multiple strings may be created with the same $data_source, but different values of $instance. The actual S4PM string is installed in this directory:

$s4pm_root/$data_source/$instance

Originally, instance was interpreted as a "gear" that enabled a data source ($data_source) to be subdivided up into forward processing and reprocessing (where gear would be set to 'forward' or 'reprocessing'). Now, $instance is a more generic interpretation in that it represents any sub flavor of a data source including simple forward and reprocessing.

Example:

```
$instance = "reprocessing";
```

9.6 $host

This parameter is MANDATORY.

The $host parameter is the name of the host machine the string runs on.

Example:

```
$host = 'g0spg11';
```

9.7 $algorithm_root

This parameter is OPTIONAL.

The $algorithm_root parameter specifies the root directory under which algorithms for this string are installed. Below this root, S4PM assumes that there is a subdirectory for each algorithm that has the name of the algorithm. Below each algorithm directory, S4PM assumes there is a version subdirectory that has the same name as the version.

For example:

```
$algorithm_root/MoPGE01/4.5.2/;
```

If the algorithm root directory is global across all strings, this variable may be set in the host or global configuration file.

If not set, the default is:

$s4pm_root/$data_source/pge

Example:

```
$algorithm_root = "/home/s4pmuser/algorithms";
```

9.8 @run_sorted_algorithms

This parameter is MANDATORY.

The @run_sorted_algorithms parameter is an array that sets the algorithms to run in this string as well as their run order in the Run Algorithm station such that the first algorithm in the list will be the one to run first if there is a choice. Stationmaster by default selects the next job to run (when a slot is available) by simple shell order.

In general, to avoid algorithm starvation, it is best to give the most upstream algorithms the highest order of preference.

Example:

```
@run_sorted_algorithms =
('GdPGE02B','MoPGE03','MoPGE02','MoPGE01');
```

9.9 @display_sorted_algorithms

This parameter is OPTIONAL.

While @run_sorted_algorithms is for sorting the priority of jobs for Stationmaster, the @display_sorted_algorithms parameter sets the order the jobs should be displayed in the S4PM Monitor. The default is to reverse @run_sorted_algorithms.

Example:

```
@display_sorted_algorithms =
('GdPGE02B','MoPGE03','MoPGE02','MoPGE01');
```

9.10 %algorithm_versions

This parameter is MANDATORY.

The %algorithm_versions parameter is a hash that lists the algorithm versions to run in this string. The hash keys are algorithm names (assumed to be listed in @run_sorted_algorithms; Section 9.8) and the hash values are their versions. Algorithms are assumed to be located in the directory specified by $algorithm_root in the host configuration file or in the default location.

> → **NOTE:** Only those algorithms listed in the @run_sorted_algorithms array are actually run in this string regardless of what is in the %algorithm_versions or %algorithm_profiles (Section 9.11) hash.

Example:

```
%algorithm_versions = (
   'MoPGE01' => '4.1.12',
   'MoPGE71' => '4.0.2',
   'MoPGE02' => '4.3.0',
   'MoPGE03' => '4.3.0',
);
```

9.11 %algorithm_profiles

This parameter is MANDATORY.

The %algorithm_profiles parameter is a hash that lists the algorithm profiles to be run in this string. The hash keys are algorithm names (assumed to be listed in @run_sorted_algorithms; see Section 9.8) and the hash values are their profiles (profiles are a subdivision of version). Algorithms are assumed to be located in the directory specified by $algorithm_root in the host configuration file or in the default location.

> → **NOTE:** The profile set in this hash must match the profile portion of the algorithm configuration file name.
>
> → **NOTE:** Also remember that only those algorithms listed in the @run_sorted_algorithms array are actually run in this string regardless of what is in the %algorithm_profiles or %algorithm_versions (Section 9.10) hash.

Example:

```
%algorithm_profiles = (
   'MoPGE01' => 'RPROC',
   'MoPGE71' => 'RPROC',
   'MoPGE02' => 'RPROC',
   'MoPGE03' => 'RPROC',
);
```

9.12 %pool_capacity

This parameter is MANDATORY.

The %pool_capacity parameter is a hash that determines the storage capacity of the disk pools that are set up for data. The hash keys are data types and the hash values are the maximum number of files (not size in bytes) for which the capacity must be set. Using the maximum number of files provided here and the maximum file size in bytes for each data type specified in the s4pm_stringmaker_datatypes.cfg file, Stringmaker will determine the sizes of each disk pool in bytes.

All data types (input and output) to be used in a string need to be specified here.

Example:

```
%pool_capacity = (
   'MoPGE01' => 100,
   'MoPGE02' => 170,
   'MoPGE03' => 150,
);
```

9.13 $config_files{'repeat_daily/s4pm_delete_expired_data.cfg'}{'%AgeLimits'}

This construct is OPTIONAL, but if not used, the parameter $data_expiration_max_hours (Section 9.14) becomes MANDATORY.

This construct is used to specify the maximum age of data files beyond which S4PM will delete them. Normally, S4PM deletes data when it knows that nothing else will need to access it. S4PM keeps track of the maximum number of uses each data file will have and it decrements the number of outstanding uses each time the file is used. When the number of outstanding uses reaches zero, the file is deleted. Sometimes, however, a file may not get used the number of times anticipated. That is where this hash comes in. The hash keys are data types and the hash values are the number of hours beyond which to delete the data regardless of any outstanding uses.

Data types not specified in this hash risk building up over time and, potentially, filling up its disk pool beyond capacity. When this happens, processing in S4PM will likely grind to a halt unless manual intervention is taken.

Example:

```
$config_files{'repeat_daily/s4pm_delete_expired_data.cfg'}{'%AgeLi
mits'} = {
   'MOD01' => 8,
   'MOD03' => 8,
   'MOD021KM' => 8,
   'MOD02HKM' => 8,
   'MOD02QKM' => 8,
   'MOD02OBC' => 8,
};
```

9.14 $data_expiration_max_hours

This parameter is OPTIONAL (but see Section 9.13).

The $data_expiration_max_hours parameter is the number of hours after which any data still resident within S4PM will be deleted from disk. Normally, data are deleted after all outstanding uses for that data file have been used up (the outstanding uses falls to zero). But the uses for some data may not fall to zero due to the production rules for optional input (*e.g.* an optional data input may show up, but after the algorithm has already given up on it). Thus, this parameter guarantees that data files won't build up indefinitely and choke the system.

The value specified here will apply to **ALL** data types. If you want different values for different data types, then see Section 9.13.

9.15 $stations{$station_name}{'$cfg_max_jobtime'}

This structure is OPTIONAL.

Optionally, for one or more stations you may specify maximum job times in seconds for certain jobs running in that station. When a running job exceeds the maximum time, the color of the box will change from green to yellow. This serves as a clue to operators that there may be a problem to investigate. There is no other effect beyond the color change. Hash keys are work order types for that station and hash values are the maximum number of seconds. By default, no maximum time is configured.

For example:

```
$stations{'find_data'}{'$cfg_max_jobtime'} = {
   'MOREDATA_MoPGE01' => 2000,
   'MOREDATA_GdPGE02B' => 150,
   'MOREDATA_MoPGE03' => 1100,
   'MOREDATA_MoPGE71' => 900,
   'MOREDATA_GdMOD02SS' => 90,
};
```

9.16 %proxy_esdts

This parameter is OPTIONAL.

The %proxy_esdts parameter is a hash that is only applicable for on-demand processing (when $on_demand is set to non zero in the string configuration file) and even then, it is optional. The %proxy_esdts hash provides a mechanism for mapping proxy data types to actual data types (aka ESDTs). In on-demand processing, very often algorithms can perform processing (e.g. subsetting) on any one of several to many data types. The easiest way to approach this situation is to tell S4PM that a proxy data type will be used to represent any one of the actual data types the algorithm will process. When this is done, the algorithm need only be configured to work with one data type, the proxy, rather than with a large list of data types.

Hash keys are the data type proxy names (which can be arbitrary) and the hash values are lists of regular expression patterns that will match the data types the proxy represents.
As yet, proxy data types cannot be used in upstream processing.

For example:

```
%proxy_esdts = (
   'MODOCL23' => ['M[OY][013AD][246OPS][1278MQWCFNS][WDAMB1]'],
   'MOD03'         => ['M[OY]D03'],
   'AIRL2CRS'   => ['AIRI2CCF','AIRX2RET','AIRX2SUP'],
);
```

9.17 S4PM Variance Parameters

The following Stringmaker host configuration file parameters have in common that they modify in some way the configuration of the S4PM string.

9.17.1 Variance Toggle Switches

In the following cases, setting the parameter to non-zero enables the option; setting it to zero or not setting it at all disables the option. Note that enabling some of these parameters may require other parameters to be set as well. Such situations are indicated below.

Parameter	Description

Parameter	Description
$has_qc	Turns on or off quality control (QC) checking of files produced in Run Algorithm. This affects all QC checking in the string. The particular QC checks performed (or not) are determined by the %qc_output hash in the s4pm_stringmaker_datatypes.cfg file.
$export_ph	Turns on or off the exporting of the production history (PH) tar files to the ECS archive (for those algorithms producing a PH file). PH files are treated somewhat like output data files. When this parameter is disabled, PH files will build up in the PH disk pool. A pseudo-cron job in the Repeat Daily station will clean up these PH files after the age out. When this parameter is enabled, the PH files will be cleaned out once a successful PAN has been received (same manner as with other exported files).
$use_checksum	Turns on or off one particular type of QC checking: the computation of a checksum for the data file (although check summing is not normally considered a QC check, it is included in S4PM QC checking for convenience). When turned on, computed checksums are included in the Product Delivery Records (PDRs) used for exporting the data to the ECS.
$has_auto_request	Turns on or off auto request functionality. When enabled, the Auto Request station is added to the string. This station and the associated Auto Request tool provide some automation for requesting data from the archive to initiate processing.
$on_demand	Turns on or off on-demand processing. When enabled, a number of stations are disabled and others are enabled. On-demand processing allows processing to be somewhat event driven (as opposed to data driven) where events (typically, requests via a user client) are sent to S4PM in the form of ODL files.
$dme	Turns on or off all data mining in S4PM. When enabled, the $sub_request_email and $pickup_dir must also be set in this file.
$data_source_polling	Turns on or off polling of input data from a disk resource rather than from the ECS archive (subscription or ordering). If set, the parameter $data_source_polling_dir must also be set.
@datapool_insert_datatypes	By including any data types in this array, this enables the insert of these data types to the ECS data pool rather than the ECS archive. When so enabled, the Insert Datapool station becomes enabled in the string. When used, the $datapool_staging_dir parameter (next) must also be set in this file.
$input_symlink_root	Turns on the symbolic linking of input data to the INPUT disk pool rather than having it pushed there. This feature is intended for situations where the input data for a string reside on a local disk, but outside of S4PM. An example is the ECS Datapool where data is ordered via

Parameter	Description
	FTP pull using the Synergy 4 distribution path. In this particular case, the data are staged by ECS to the Datapool. Using this parameter, S4PM can be configured to create symbolic links to those data (assuming they're visible locally). Set the parameter to the machine and root path (separated by a colon) of the data location on the local machine. The directory specified in the DN will be appended to this root path to find the data. Example: $input_symlink_root = "g0dps01:/data/root/"; NOTE: This parameter requires that the $s4pm_email parameter be set in the Stringmaker Global configuration file.

Table 9-1. Stringmaker host configuration file parameter switches that alter the behavior of S4PM.

9.17.2 Other Variance Parameters

The following parameters are dependent upon the one above.

Parameter	Description
$datapool_staging_dir	This parameter is required if any data types are listed in the @datapool_insert_datatypes array (see above). This parameter sets the staging directory location on the machine that runs the script to insert data files into the ECS data pool action queue. The directory must be visible on that machine.
$sub_request_email	This parameter is required if $dme is set to non-zero. The $sub_request_email should be set to the e-mail address of the user responsible for setting up subscriptions in ECS. The '@' symbol needs to be escaped. Example: $sub_request_email = "help\@daac.gsfc.nasa.gov";
$pickup_dir	This parameter is required if $dme is set to non-zero. The $pickup_dir should be set to the directory on ECS data pool where output products will be placed. Typically, this directory is in an anonymous FTP area where a Data Mining Edition user can retrieve the data. The directory needs to be local for that machine.
$scli_host	If SCLI is not installed on the machine where S4PM is to be run, this variable should be set to the machine where SCLI is to be accessed remotely using secure shell. If SCLI is installed locally, this variable should be set to the empty string or unset.
$data_source_polling_dir	This parameter must be set if $data_source_polling is set to non-zero. The $data_source_polling must be set to the root of the polling directory. The root directory is that directory under which the category directories exist (e.g. MOAT, MOOG) as configured in ECS data pool. It is under these directories that the data type subdirectories exist named: <datatype>:<versioned> For example: MOD08_M3.004 And under the data type directories are directories for each date (YYYY.MM.DD as in 2004.11.27). Finally, under the date directories, the data and XML files are assumed to reside. Thus, $data_source_polling_dir/<category>/<datatype>/<date>/data In theory, any data area structured similar to the ECS data pool can be used with this option.

Table 9-2. Stringmaker host configuration file parameters that depend upon the parameters shown in Table 9-1.

9.18 %ordering_tool_parms

This parameter is OPTIONAL.

The %ordering_tool_parms parameter is a hash that configures the Ordering interface available in the Request Data station. There are two attributes. The first is increment which sets the width (in seconds) of the smallest interval in the Compose Data request tool. The default is 7200 seconds which means that the display will show a day divided up into 12 two-hour increments.

The second attribute is files_per_hour which affects the Fill Hole ordering interface. In this tool, the total width is one 'increment' and it is sub-divided up into files_per_hour sub-increments. The default is 12 meaning that each sub-increment is 300 seconds (if 'increment' is 7200).

For example:

```
%ordering_tool_parms = (
    'increment'        => 7200,
    'files_per_hour' => 12,
);
```

9.19 Parameter Overrides

The Stringmaker string configuration files are often the files in which earlier defined parameters can be overridden. For example, the data type SEA_ICE version may be specified in the s4pm_stringmaker_datatypes.cfg file as '003'. In one string, however, you need to use version '004' of this data type without affecting the '003' version used in all other strings. The easiest way to do this is to put the following override statement in the <string>.cfg file:

```
$all_datatype_versions{'SEA_ICE'} = '003';
```

For this one string, the version '003' trumps the '004' in the Stringmaker data types configuration file.

10. The Stringmaker Algorithm Configuration File

There must be at least one algorithm configuration file for each version of each algorithm.

> → **NOTE:** Starting with S4PM 5.6.2, the format of the algorithm configuration files has changed with the introduction of Stringmaker, a replacement for Stringmaster. When using Stringmaker, the new format must be used. Since Stringmaster will be phased out, you are strongly encouraged to use Stringmaker and therefore, this new algorithm configuration format.

Unlike with the Stringmaster algorithm configuration file, the Stringmaker algorithm configuration file is used for both configuring a string and for running the string. Another distinction between the two formats is that the Stringmaster algorithm configuration file was in a parameter=value format whereas the Stringmaker algorithm configuration file is in Perl syntax, like most other S4PM configuration files.

10.1 File Name

The file name for the algorithm Stringmaker configuration file must be:

<algorithm_name>_<profile_name>.cfg

where <algorithm_name> is the name of the algorithm and <profile_name> is the name of the profile for this algorithm. A profile allows the same algorithm to have more than one set of production rules.

For example:

MoPGE02_RPROC.cfg

10.1.1 Mandatory Parameters

A number of parameters are mandatory. The sections below describe these parameters.

10.1.1.1 $algorithm_name

This parameter is MANDATORY.

The $algorithm_name parameter is a string representing the algorithm name. It must match the name of the directory into which the algorithm is installed and the $algorithm_name parameter set in the Stringmaker algorithm configuration file name.

Example:

```
$algorithm_name = 'MoPGE01';
```

10.1.1.2 $algorithm_version

This parameter is MANDATORY.

The $algorithm_version parameter is a string representing the algorithm version. It must match the name of the subdirectory under the algorithm directory into which the algorithm is installed.

Example:

```
$algorithm_version = '2.4.3m';
```

10.1.1.3 $algorithm_exec

This parameter is MANDATORY.

The $algorithm_exec is a string representing the name of the executable to run for this algorithm. It may be a binary executable, script, or a wrapper script calling other scripts or binaries. There can only be one value for this parameter.

Example:

```
$algorithm_exec = ‘PGE02.csh’;
```

10.1.1.4 $processing_period

This parameter is MANDATORY.

The $processing_period parameter is the processing period in seconds. It specifies over what ***data*** time length the algorithm is to run (***not*** wall-clock time!). If the processing period is less than the time coverage specified for the trigger input data (see Section

10.1.1.11), multiple MOREDATA work orders will be produced (each resulting in a algorithm run) spanning the processing period.

> → **NOTE:** The start time of any particular algorithm run is based upon the start time of a particular trigger data file. To make the processing start time independent of the trigger data start time, use the $processing_start parameter (Section 10.1.2.2).

If the $processing_period is set to zero and the $trigger_coverage (Section 10.1.1.7) is set to zero, the number of work orders output and the start and end times of the processing period written into the output work orders are determined by the start and end times of the data in the NEWDATA input work order and the $product_coverage (Section 10.1.1.5). This is useful for production where the time coverage of the trigger data is not fixed but the output product coverage is (for example, in direct broadcast).

Another production rule is triggered by setting both $processing_period and $trigger_coverage to zero. In this case, the correct number of output work orders and the processing start and stop times in those work orders will accommodate the trigger input data coverages dynamically.

In summary:

Settings	Production Rule Result
$processing_period = $trigger_coverage	One run per trigger data file.
$processing_period < $trigger_coverage	Multiple number (fixed) runs per trigger data file (equal to $trigger_coverage/$processing_period)
$processing_period = 0 **and** $trigger_coverage = 0	Multiple number (dynamic) runs per trigger data file enough to cover the particular input with data time aligned with dynamic input.

Table 10-1. Possible settings of the $processing_period and $trigger_coverage parameters and the resultant production rule invoked.

Example:

```
$processing_period = 300;
```

10.1.1.5 $product_coverage

This parameter is MANDATORY.

The $product_coverage parameter is the time coverage (in seconds) of the output products. Note that the assumption here is that ALL output products from a algorithm have the same time coverage. Although this attribute is somewhat redundant, since the

coverages of individual data types are already contained in this configuration file, it does have a special purpose.

In the case where the trigger input data coverages are not fixed in length (for example, direct broadcast), the $product_coverage is used along with the start and stop times of the input NEWDATA work order to dynamically determine the number of work orders to output and the processing start and stop times in those work orders. Setting the $processing_period and the $trigger_coverage to zero triggers this feature. See Table 10-1.

Example:

```
$product_coverage = 300;
```

10.1.1.6 $metadata_from_metfile

This parameter is OPTIONAL.

For algorithms that employ the ECS Toolkit, the $metadata_from_metfile parameter simply indicates whether metadata reads (of input data) should be from the accompanying metadata files or from the files themselves which are assumed to be HDF. The choice affects how the runtime PCFs are generated. Note, this setting applies to ALL input products. The valid choices are 0 (all reads will be from HDF files) and 1 (all reads will be from accompanying metadata files.

This parameter has no effect on algorithms that do not use the ECS Toolkit or for input files that are not in HDF.

Example:

```
$metadata_from_metfile = 0 ;
```

10.1.1.7 $trigger_coverage

This parameter is MANDATORY.

The $trigger_coverage parameter is the time coverage of the trigger input in seconds. Normally, this should match exactly the coverage for the trigger data type (see %inputs, %outputs).

In on-demand processing, the $trigger_coverage associated with the PSPEC trigger input is ignored.

Note that the $trigger_coverage in conjunction with the $processing_period can be used to invoke several production rules. See Table 10-1.

Example:

```
$trigger_coverage = 7200;
```

10.1.1.8 $pcf_path

This parameter is MANDATORY.

The $pcf_path parameter is the full or relative path to the SDP Toolkit's Process Control File (PCF) template for this particular algorithm. This PCF template will be the basis for generating the runtime PCFs for each algorithm run.

Example:

```
$pcf_path = "../prepare_run/GDAAC.PGE01.pcf.tpl ";
```

10.1.1.9 @stats_datatypes

This parameter is MANDATORY.

The @stats_datatypes parameter is an array that contains a list of output data types on whom performance statistics are to be generated in the Run Algorithm station. Not all data types necessarily need to be listed here. But it is simpler to list them all.

Example:

```
@stats_datatypes = ('MOD01', 'MOD03');
```

10.1.1.10 $stats_index_datatype

This parameter is MANDATORY.

The $stats_index_datatype parameter is set to the one data type listed in the @stats_datatypes (Section 10.1.1.9) considered to be the index or main data type.

Example:

```
$stats_index_datatype = 'MOD01';
```

10.1.1.11 %inputs, %outputs

These parameters are MANDATORY.

The %inputs hash describes dynamic inputs data used by the algorithm and the %outputs hash describes dynamic output data generated by the algorithm. Static input files are assumed to permanently reside with the algorithm and are fixed in the algorithm's PCF template file. All possible input data and all possible output data must be described in these hashes.

The hash keys are unique tags such as input1, input2 and output1, output2. They can be any string as long as they are unique within their respective hashes.

For each such key, a number of attributes and their values describe various aspects of the input and output data. These attributes are described in the following table:

Attribute	Description
data_type	The data type name (ESDT ShortName if in ECS) of the input or output. May be a proxy data type (see 10.1.1.11.1)
data_version	The data type version (ESDT VersionID if in ECS) of the input or output. May be for a proxy data type. See 10.1.1.11.2.
need	For inputs, this sets the need; for outputs, it is generally ignored, but can be used to set spatial region identifiers. See Section 10.1.1.11.3. Valids are: REQ, REQn, TRIG, OPTn, Spatial_Tag.
timer	Input wait timer in seconds. Ignored for outputs. See 10.1.1.11.4.
lun	PCF logical unit number (LUN). See Section 10.1.1.11.5.
currency	The input currency. Valids are: CURR, PREVn, FOLLn, NPREVn, NFOLLn. See 10.1.1.11.6.
coverage	The time coverage in seconds of the input. Ignored for output. See Section 10.1.1.11.7.
boundary	Data boundary. See Section 10.1.1.11.8.

Table 10-2. Hash attributes of the %inputs and %outputs hashes in the Stringmaker algorithm configuration file.

Below is an example for an algorithm that produces one output from two inputs:

```
%inputs = (
    'input1' => {
        'data_type' => 'MOD01',
        'data_version' => '005',
        'need' => 'TRIG',
        'timer' => 0,
        'lun' => '79901',
        'currency' => 'CURR',
        'coverage' => 300,
        'boundary' => 'START_OF_DAY',
    },
   'input2' => {
        'data_type' => 'MOD03',
        'data_version' => '005',
        'need' => 'REQ',
        'timer' => 0,
        'lun' => '79920',
        'currency' => 'CURR',
        'coverage' => 300,
        'boundary' => 'START_OF_DAY',
    },
);
%outputs = (
    'output1' => {
        'data_type' => 'MOD02',
        'data_version' => '005',
        'lun' => '79901',
        'currency' => 'CURR',
        'coverage' => 300,
    },
);
```

10.1.1.11.1 data_type

The data_type attribute is needed for both input and output entries. It is the data type name (ESDT ShortName if in ECS). If the data type is to be from the ECS archive, there must be a valid ESDT descriptor file for this data type and version installed and configured in the ECS.

For on-demand processing, the output data types do not get archived and, therefore, there is no need for a valid ESDT descriptor file in the ECS.

10.1.1.11.2 data_version

The data_version attribute is needed for both input and output entries and is the data type version (VersionID associated with the ESDT ShortName in ECS).

10.1.1.11.3 need

The need attribute expresses whether the input is required or optional and to what degree. This field is ignored for output files except in the case where it is used to tag spatial regions (see Section 10.1.2.11). The following table lists the possible settings for the need attribute and their meanings:

need Setting	Descriptions
REQ, REQ*n*	The input is required; the algorithm cannot run without it. The *n* is an integer expressing the order of preference of the input relative to others having the same LUN (with 1 being the most preferred). Note that REQ is equivalent to REQ1. This alternate input production rule is used in the case where some data file for this LUN is required.
TRIG	Same as REQ (which is the same as REQ1), but this marks the input as the data type that triggers the algorithm. The trigger input MUST be the one set for DATA_TYPE_TRIGGER and MUST be the first input in this configuration file. Multiple algorithms may use the same data type as a trigger.
OPT*n*	The input is optional; the algorithm will run without it. The *n* is an integer expressing the order of preference of the input relative to others having the same LUN (with 1 being the most preferred). Unlike with REQ*n*, if none of the data files for this LUN are found, the algorithm will still be run.

Table 10-3. Possible setting of the need attribute in the %inputs hash in the Stringmaker algorithm configuration file.

10.1.1.11.4 timer

The timer attribute is the timer in seconds that represents how long the production system should wait for the input. The timer for required input (anything with a need of REQ*n*) starts once the trigger input arrives. The timer for optional input (anything with a need of OPT*n*) starts once all of the required input arrives. The timer is ignored for the trigger data type and for all outputs.

10.1.1.11.5 lun

This is the logical unit number (PCF) associated with this data type as listed in the PCF template.

10.1.1.11.6 currency

The currency expresses how the input data is aligned with the trigger data file in terms of start and stop times. This field is ignored for output files. Valid settings are described in the table below:

currency Setting	Description
CURR	The input is contemporaneous with the algorithm processing period
PREV*n*	The input is *n* steps previous to the algorithm processing period (by increments equal to the time coverage of the input itself). A PREV1 means the previous data file, a PREV2 means the data file before that, etc.
FOLL*n*	The input is *n* steps following the algorithm processing period (by increments equal to the time coverage of the input itself). A FOLL1 means the following data file, a FOLL2 means the data file following that, etc.
NPREV*m,n*	The input requested is the nearest in time (looking backward) to the processing period of the algorithm. The *n* and *m* are integers that specify how far to begin looking back and how far to look back. *m* is an integer that specifies where to begin looking back with 0 being the current time period (equivalent to CURR). *n* is an integer that specifies the last time period to look back. For example, 'NPREV0,4' means look first for the current data file (that's what the 0 means). If that is not available, then look for the previous one (equivalent to PREV1). If that is not available, then look for the one previous to that (equivalent to PREV2). And so on with the equivalent to PREV4 being that last one. To do the same thing but omit the current data file in the search, use 'NPREV1,4' instead. The Select Data station will, in fact, convert NPREV*m,n* into the appropriate CURR and/or PREV*n* equivalents. The timer associated with each equivalent entry will be the original timer split evenly among the equivalents. Thus, if we used 'NPREV0,3' and a timer of 7200, the current data file would be searched for up to 1800 seconds. If it wasn't found within that time period, the search for the next, PREV1, would begin and expire 1800 seconds later. If that data file wasn't found, the search for PREV2 would commence and so on.
NFOLL*m,n*	This functions the same as NPREV*m,n* described above, except the input requested is the nearest in time looking forward.

Table 10-4. Possible setting of the currency attribute in the %inputs hash in the Stringmaker algorithm configuration file.

10.1.1.11.7 coverage

The coverage attribute is the temporal coverage of the data in seconds.

10.1.1.11.8 boundary

The boundary attribute is the data boundary against which to determine start times of input data files. Valids are START_OF_WEEK, START_OF_DAY, START_OF_12HOUR, START_OF_8HOUR, START_OF_6HOUR, START_OF_4HOUR, START_OF_2HOUR, START_OF_HOUR, START_OF_MIN, and START_OF_SEC. An offset in seconds, plus or minus, may be applied to the

boundary (e.g. START_OF_DAY-3600 to make the boundary be 23:00 hours rather than 00:00 hours). This attribute is ignored for output files.

10.1.1.12 %input_uses, %output_uses

These parameters are MANDATORY.

These two hashes specify the number of times each data type is used by this algorithm. The hash keys are the data type names and the hash values are the number of uses for that data type. For outputs, the number of uses is almost always set to 1 (the export of a product is considered a use). For inputs, consider how many runs of the algorithm will use that input. Typically, it is 1. But for inputs spanning a long range in time, several runs of the algorithm may be needed to process the entire file.

Example:

```
%inputs_uses = (
    'MOD000' => 9,
    'AM1ATTN0' => 9,
    'AM1EPHN0' => 9,
);

%output_uses = (
    'MOD01' => 1,
    'MOD03' => 1,
);
```

10.1.2 Optional Parameters

The following parameters are optional in that they are only needed if the particular functionality is desired.

10.1.2.1 $post_processing_offset, $pre_processing_offset

These parameters are OPTIONAL.

By default, the beginning of the algorithm processing period is aligned with the start time of the input trigger data file. An offset from that alignment can be specified here as positive or negative seconds. If positive, the processing period will start after the trigger data file time by the amount specified. If negative, the processing period will start before the trigger data file time by the same amount. The default is zero if not specified.

With $post_processing_offset, the offset is applied in a post examination sense. That is, the Select Data does its determination of data times relative to the processing period assuming no offset (e.g. the definition of current or previous data file is based on no

offset). Only at the point where the processing start and stop times are written into the output PDR is the processing offset applied.

With $pre_processing_offset, the offset is applied in a pre examination sense. That is, the Select Data station does its determination of data times relative to the processing period assuming this offset (e.g. the definition of current or previous data file is based on this offset).

Examples:

```
$pre_processing_offset = 300;

$post_processing_offset = 600 ;
```

10.1.2.2 $processing_start

This parameter is OPTIONAL.

By default, the processing start time is aligned to the start time of the trigger data file (with PRE_ or POST_PROCESSING_OFFSET applied). To make the processing start time completely independent of the start time of the trigger data, use PROCESSING_START. Valids are the same as is used for data boundary, for example: START_OF_WEEK, START_OF_DAY, START_OF_6HOUR, START_OF_HOUR, START_OF_MIN, and START_OF_SEC. Unlike data boundaries, however, offsets of plus or minus cannot be added to these.

10.1.2.3 $make_ph

This parameter is OPTIONAL.

The $make_ph parameter enables or disables the generation of a production history (PH) tar file associated with every run of the algorithm. The PH tar file contains logs and other information about the run that may be useful in debugging an algorithm.

To enable PH generation, set this parameter to a non-zero value. To disable, set it to zero or leave it unset. The default is to not produce a PH file.

> → **NOTE:** Enabling PH generation doesn't necessarily mean that the PH will be exported to the archive. That is controlled by the $export_ph parameter in the Stringmaker string-specific configuration file.

Example:

```
$make_ph = 1;
```

10.1.2.4 $apply_leapsec_correction

This parameter is OPTIONAL.

The $apply_leapsec_correction parameter indicates whether the leap second and AIRS instrument offset corrections should be applied to the process start and stop times (LUNs 10258 and 10259) in the runtime PCF. If set to zero, no leap second or instrument offset corrections are applied to the start and stop times. If set to non-zero, the leap second and instrument offset corrections are applied to the start and stop times in the runtime PCF. The default is 0 (disabled).

This option does NOT affect the data start and stop times by which Find Data will search for data. For that, see $leapsec_datatypes.

> → **NOTE:** This parameter is pertinent only to AIRS data processing and will likely be removed from the S4PM baseline.

Example:

```
$apply_leapsec_correction = 1;
```

10.1.2.5 $leapsec_datatypes

This parameter is OPTIONAL.

This parameter is a comma or space delimited list of data types in which the leap second and AIRS instrument offset corrections should be applied. This affects the data times and thus the file name patterns that Find Data will use to search for inputs. This option does NOT affect the process start and stop times in the PCF (LUNs 10258 and 10259). For that, see $apply_leapsec_correction.

> → **NOTE:** This parameter is pertinent only to AIRS data processing and will likely be removed from the S4PM baseline.

Example:

```
$leapsec_datatypes = "AIRIASCI AIRIACAL AIRIBRAD";
```

10.1.2.6 $algorithm_station

This parameter is OPTIONAL.

This parameter is used rarely. Normally, all algorithms run within the single Run Algorithm station. There is the option, however, to have one or more algorithms run in other stations that have a different name than 'Run Algorithm'. This parameter sets that name. Other than the name of the station, there is no functional difference between it and the Run Algorithm station. The value specified in this parameter will become the station directory name of the alternate Run Algorithm station. The S4PM Monitor will display these extra Run Algorithm stations.

Example:

```
$algorithm_station = 'run_special_algorithm';
```

10.1.2.7 %specialized_criteria

This parameter is OPTIONAL.

The %specialized_criteria parameter option is only used for on-demand processing and only for algorithms that need to have runtime parameters passed from the user's client, through the V0 Gateway and into the runtime PCF. On-demand algorithms that read the PSPEC file directly do not need this parameter set.

For each of the specialized criteria from the request ODL that are to appear in the runtime PCF, a hash key-value pair must be added to the %specialized_criteria hash as illustrated below:

```
%specialized_criteria = (
    '21200' => 'FORMAT|MOD021KM.005, MOD02HKM.005, MOD02QKM.005',
    '21210' => 'CHANNELS|MOD021KM.005, MOD02HKM.005,
MOD02QKM.005',
);
```

The hash keys are simply the PCF LUNs in which the runtime parameter will be placed. The hash values have two parts separated by a pipe (|) character. The first part is the specialized_criterion_name, which must match exactly the specialized criterion name as it appears in the request ODL file and the second part is a list of data type and version against which this specialized criterion applies.

In the above example, the runtime PCF will contain an entry for LUN 21200. That runtime parameter will be named 'FORMAT' and the value contained in LUN 21200 will be what ever was contained in the request ODL specialized criterion named 'FORMAT' (e.g. a format specification) if the data type was one of the ones listed. The PCF will also contain LUN 21200 with the name 'CHANNELS' and it will contain, presumably, a list of channels.

The list of data types (and versions) is necessary since different data types might have different specialized criteria associated with them.

10.1.2.8 %file_accumulation_parms

This parameter is OPTIONAL.

The %file_accumulation_parms parameter is a hash that is used for algorithms that invoke the file accumulation production rule. This production rule can be used when there are many files of a particular data type needed as input to a single run of the algorithm and is particularly useful if that data type is the trigger data type.

Normally, each arriving trigger file will result in a separate run of the algorithm. Using the file accumulation production rule, however, the arriving data files will accumulate to a specified number and only then trigger a single run of the algorithm on the set of accumulated files.

When invoked, the file accumulation production puts the Select Data station into another mode where it polls for the data needed. Normally, polling for data is the job of Find Data station, but Select Data polls at a far lower frequency. Once sufficient data have been located, Select Data reverts to its normal mode of operation and determines what other data are needed by the algorithm. Then, the work order is passed to the Find Data station as normal.

The %file_accumulation_parms hash is a consolidation of a recipe of steps that had to be set individually in previous releases of S4PM. This complex and error-prone recipe is still supported but should be considered deprecated.

The follow table describes the attributes in the %file_accumulation_parms hash:

Attribute	Description
window_width	This sets the width of the accumulation window in seconds, the time period over which data are to be accumulated.
window_boundary	This sets the boundary against which to align the accumulation window itself. Valids are the same as are available for the 'boundary' attribute of the %inputs hash.
polling_interval	Sets how often in seconds the data to be accumulated are polled for.
timer	Sets the maximum amount of time to wait in seconds for all files to accumulate.
file_threshold	Sets the minimum number of files needed by the algorithm. If the file_threshold has been met by the time the timer is up, the job will succeed. If that minimum hasn't been met, the job will fail.

Table 10-5. Hash attributes of the %file_accumulation_parms hash in the Stringmaker algorithm configuration file.

For example:

```
%file_accumulation_parms = (
    'window_width' => 86400,
    'window_boundary' => ' START_OF_DAY',
    'polling_interval' => 7200,
    'file_threshold' => 10,
    'timer' => 86400*3,
);
```

10.1.2.9 $preselect_data_args

This parameter is OPTIONAL.

The $preselect_data_args parameter specifies the arguments that are to be passed to the s4pm_preselect_data.pl script running in the Select Data station. This was a necessary step to invoke the file accumulation production rule.

The %file_accumulation_parms parameter, however, makes this parameter obsolete.

If you choose to use the $preselect_data_args, the arguments to specify are the polling interval with the -i option, the file threshold with the -thresh argument, and the timer with the -timer argument. For definitions of these attributes, see B.1.2.8.

For example:

```
$preselect_data_args = '-i 7200 - thresh 10 -timer 86400';
```

10.1.2.10 $trigger_block_args

This parameter is OPTIONAL.

The $trigger_block_args specifies the command to implement blocking in S4PM. A block prevents new data showing up in S4PM (vi the Register Data station) from triggering a new run of an algorithm for which they are associated. A block is defined over a particular time interval such that data arriving whose times are outside of that interval are allowed to trigger new algorithm runs, but those occurring within the interval are quietly removed.

Once a block is defined for a particular time interval, that block is not actually created into the first data falling within that interval arrives. This first arriving data is allowed to trigger an algorithm run, but all subsequent ones will be blocked.

Blocks are typically used with the file accumulation production rule where the trigger data type is the data type to be accumulated. In such a case, you only want one of the data types within an interval to trigger a run, not all of them.

The %file_accumulation_parms hash parameter makes this parameter obsolete for this purpose as it already handles the blocking implicitly.

10.1.2.11 Spatial Identifiers

This feature is OPTIONAL.

In S4PM, data files are named using the data time and the production time (see Section 6.5.1).. This leads to the question of how an algorithm might produce multiple distinct data sets that share the same temporal coverage and therefore, the same file name. Unless S4PM can make each file name distinct, one file will overwrite the other.

The answer to the above problem is to allow, for this unique situation, a way to modify the file name with something unique for each such file. This is accomplished in S4PM using the need attribute of the %outputs hash which, for output files, is normally not used. If S4PM detects a value for need in the %outputs hash, it interprets this value as something to include in the output file name. Thus, this normally unused attribute is co-opted for this use.

The value specified for the need, in this situation, can be any string that will be made part of the output file name (after the data type). A unique tag in this field must be associated with each unique output LUN having the same data type name and version (ESDT ShortName and VersionID in ECS). The file names referred to here are those that exist in the S4PM file system.

The following constraints apply for spatial subsetting:

1. All spatial subsets must use the same data type name and version.
2. Each subset must use a unique PCF LUN and all possible LUNs must appear in the PCF template. Typically, each corresponds to a unique region.
3. For now at least, the subsetted products cannot be used as input to downstream algorithms.

Below is an example of how spatial subsets may be configured in an algorithm configuration file. Here, an algorithm outputs data types MOD02SSH and MOD02SSN which are handled in the normal way. But in addition to these data, the algorithm also produces spatial subsets in data type MOD021SC. All MOD021SC data have the same temporal coverage, but are uniquely identified by the 3-character values set with the need attribute. The values in this field will become part of the file name in S4PM:

```
%outputs = (
# Regular output
    'output1' => {
        'data_type' => 'MOD02SSH',
        'data_version' => '004',
        'lun' => '22222',
        'currency' => 'CURR',
        'coverage' => 300,
    },
    'output2' => {
        'data_type' => 'MOD02SSN',
        'data_version' => '004',
        'lun' => '22225',
        'currency' => 'CURR',
        'coverage' => 300,
    },
# Spatial output
    'output3' => {
        'data_type' => 'MOD021SC',
        'data_version' => '004',
        'lun' => '30001',
        'currency' => 'CURR',
        'coverage' => 300,
        'need' => 'FLX',
    },
   'output4' => {
        'data_type' => 'MOD021SC',
        'data_version' => '005',
        'lun' => '30002',
        'currency' => 'CURR',
        'coverage' => 300,
        'need' => 'BAP',
    },
    'output5' => {
        'data_type' => 'MOD021SC',
        'data_version' => '005',
        'lun' => '30003',
        'currency' => 'CURR',
        'coverage' => 300,
        'need' => 'GTP',
    },
    'output6' => {
        'data_type' => 'MOD021SC',
        'data_version' => '005',
```

```
        'lun' => '30004',
        'currency' => 'CURR',
        'coverage' => 300,
        'need' => 'GTX',
    },
    'output7' => {
        'data_type' => 'MOD021SC',
        'data_version' => '005',
        'lun' => '30005',
        'currency' => 'CURR',
        'coverage' => 300,
        'need' => 'DCB',
    },
    'output8' => {
        'data_type' => 'MOD021SC',
        'data_version' => '005',
        'lun' => '30006',
        'currency' => 'CURR',
        'coverage' => 300,
        'need' => 'GOM',
    },
    'output9' => {
        'data_type' => 'MOD021SC',
        'data_version' => '005',
        'lun' => '30007',
        'currency' => 'CURR',
        'coverage' => 300,
        'need' => 'NOS',
    },
);
```

→ **NOTE:** All ***possible*** spatial identifiers have to be specified in the algorithm configuration file even though in any one run, there may be only a few, one, or no data produced.

This feature can be used in any situation where multiple outputs of the same data type, data version, and data time need to be produced with distinct file names. Although the example showed spatial identifiers with only three characters, any length string will work.

11. The Stringmaker Jobs Configuration File

The optional Stringmaker jobs configuration file is identical to the s4pm_max_children.cfg configuration file of Stringmaster. Its sole purpose is to specify the maximum number of jobs per station per string. Unless specified in this file, the default maximum for most stations is five.

11.1 File Name

The file name of the Stringmaker jobs configuration file is:

s4pm_stringmaker_jobs.cfg

The only parameter in the s4pm_stringmaker_jobs.cfg file is a double-keyed hash, %max_children, whose first and second keys are the string ID and station, respectively. The hash values are the maximum number of jobs to run in that station of that string.

For example:

```
$max_children{'S4PM10_MO_FW'}{'run_algorithm'} = 3;
$max_children{'S4PM10_MO_FW'}{'run_algorithm71'} = 1;
$max_children{'S4PM10_MO_FW'}{'find_data'} = 5;
$max_children{'S4PM07_AI_FW'}{'find_data'} = 72;
$max_children{'S4PM07_AI_FW'}{'run_algorithm'} = 5;
$max_children{'S4PM07_MY_FW'}{'run_algorithm'} = 6;
$max_children{'S4PM07_MY_FW'}{'run_algorithm71'} = 1;
```

The string IDs (first hash key) must match exactly the $stringid parameter as specified in a Stringmaker string configuration file for a string (see Section 9.2). The station (second key) must be the name of a station as identified by its directory name. Note that this file only needs to contain those strings and stations for which the default is not acceptable.

The stations that typically one wants to have in this file are:

- Run Algorithm (run_algorithm)
- Find Data (find_data)
- Allocate Disk (allocate_disk)

The Modify Max Children tool available from the S4PM Monitor allows one to modify "on-the-fly" the maximum number of jobs for a particular station within a particular string. In fact, this tool will update the s4pm_stringmaker_jobs.cfg file to reflect those changes (at the bottom along with a timestamp).

→ **NOTE:** There is one important caveat with this tool, however, in the current release of S4PM: The Modify Max Children tool can only modify stations and strings that are already in the s4pm_stringmaker_jobs.cfg file. If, for example, the 'run_algorithm' station for string S4PM10_AU_FW is not already in the Stringmaker jobs configuration file, the Modify Max Children tool cannot set or modify it.

12. The Stringmaker Derived Configuration File

The Stringmaker derived configuration file, like Stringmaker static configuration file, should not need to be modified. Its purpose is to be the bottom feeder among all of the other Stringmaker configuration files. Using the information specified before it on data types and algorithms and variances, s4pm_stringmaker_derived.cfg finalizes the configuration of the S4PM string. It does so by completing the configuration information of stations defined earlier by static configuration file and by building other stations from scratch.

12.1 File Name

The file name for the Stringmaker derived configuration file is:

s4pm_stringmaker_derived.cfg

Like the static configuration file, the derived configuration file is broken up into sections for each station that gets specified. The information described in Section 4.4.4 for the s4pm_stringmaker_static.cfg configuration file applies equally to the s4pm_stringmaker_derived.cfg file as well.

13. Working With Algorithms

This section discusses the heart of any S4PM string, the algorithms running within.

13.1 What Algorithms Can S4PM Support?

Essentially any algorithm code can be supported by S4PM. The following, however, are some things to consider:

1. Algorithms should not assume a particular directory structure. This means that the output file locations, input file locations, and the location from which the algorithm is running should not be hard coded into the algorithm. An algorithm that does hard code these items can be made to work in S4PM, but it requires extra work.

2. Algorithms should produce metadata files for the products they produce. The metadata format is ODL or XML using the EOSDIS data model. Algorithms that don't produce metadata will need to be wrapped by a script that carries out this function for them.

3. An algorithm that requires command line arguments can be handled easily so long as the arguments are static, that is, they don't change from one run to another. If this is not the case, a wrapper script would need to be written that finds and sets the runtime value of any dynamic arguments.

13.2 Algorithm Production Rules

S4PM supports a fairly rich set of production rules that control the inputs that each algorithm sees at runtime. A summary of the production rules supported in S4PM is:

- Basic production of one or more products having the same temporal coverage as the input.
- Time-shifted inputs forward or backward in time relative to the triggering input.
- Time-shifted processing period relative to the triggering input.
- Designation of both required and optional input.
- Multiple alternate inputs, both required and optional, with order or preference specified.
- Wait timers on all inputs (except the triggering input).
- Spatial subsetting whereby all output have the same temporal coverages, but are spatially distinct.
- Input accumulation to support daily or multi-day compositing or aggregating algorithms.
- Proxy data types that represent more than one input data type.

13.3 Production Rule Concepts

13.3.1 Simple Production Scenarios

```
INPUT A ⇒ Algorithm ⇒ OUTPUT C
```

The simplest production rule is an algorithm that reads in one data file of data type A and outputs one data file of data type C. Such an algorithm will run every time a data file of data type A arrives. If three such data files arrive at once, three separate runs of the algorithm will be kicked off in S4PM. Here, we assume that the time coverage of the output is the same as the time coverage of the input. Further, we assume each run is completely uncorrelated. Hence, if the input data type A file is has a coverage from Oct 23, 2004 10:00:00 to Oct 23, 2004 10:05:00, the time coverage of the output C will be the same. The above is a description of the most simple production rule.

```
INPUT A
                ⇒ Algorithm ⇒ OUTPUT C
INPUT B
```

A slightly more complex (and realistic) production rule is one that has more than one input. For example, data types A and B are both needed to produce one output file of data type C. In this case, a run of the algorithm will not occur until both data types A and B arrive. We can just as easily have three or more inputs. Likewise, the number of outputs is unrestricted. In fact, an algorithm may produce no output at all (for example, an algorithm that updates a database with a new table row without producing any output file).

```
          INPUT A (Trigger)
                                          ⇒    Algorithm    ⇒
OUTPUT C
INPUT B (1 Step Earlier)
```

In the above examples, we assumed that the time coverage of the output files matched that of the input. But this is not a requirement. In fact, S4PM can support the notion of time-shifted inputs. An algorithm may need one or more of its inputs shifted in time (backward or forward) relative to a data type designated as the trigger data type. For example, if we designate input A as the trigger, an algorithm may require that input B not have the same time coverage as A, but be the one earlier in time.

S4PM further supports optional inputs. An algorithm will not be run unless all of the required inputs are available. If an input data type is designated as optional, the algorithm will look for that data type, but if it cannot be found within a configurable time limit, the algorithm will run without. There can be more than one optional input and these optional inputs can be ordered as to what is the most desired through what is the least desired. S4PM will attempt to use the most desired optional input. If not available, it will attempt

to look for the next most-desired input, etc. If none of the optional inputs are available, the algorithm will run without it.

You will often see the term PGE. This simply refers to the algorithm and in this context, PGE is synonymous with algorithm. (PGE actually stood for Product Generation Executive).

13.3.2 The Stringmaker Algorithm Configuration File

The production rules illustrated above and many others are embodied in the Select Data configuration file. Once Select Data configuration is needed or each algorithm. In fact, the Select Data configuration file is part of the algorithm package (discussed below). Here, we discuss this important configuration file and how to generate it.

13.3.2.1 The Algorithm Configuration File Name

As discussed in Section 10.1, the algorithm configuration file must be named:

<algorithm_name>_<profile_name>.cfg

where:

<algorithm_name> is the name of the algorithm and <profile_name> is the name of the profile for this algorithm. Any one algorithm may have multiple profiles and therefore, multiple algorithm configuration files each with a file name distinguished by the profile name. The most likely reason to have multiple profiles is to maintain distinct sets of production rules.

Example valid algorithm configuration file names are:

MoPGE02_nominal.cfg
AiL2_reproc.cfg

13.3.2.2 Algorithm Configuration File Content

Section 10 has a full description of the parameters that go into an algorithm configuration file. The format of the algorithm configuration file is the same as all Stringmaker configuration files, namely Perl syntax.

It is always wise to verify algorithm configuration file syntax by running it through the Perl compiler:

```
perl –c <algorithm>_<profile>.cfg
```

The format of the Select Data configuration file is parameter = value. Some parameters are mandatory while others are optional and may be used only when needed by the algorithm.

13.3.3 Algorithm Configuration File Autopsy

In this section, we will discuss in detail an example algorithm configuration file. The file may be seen in Appendix A. Note that the line numbers at the beginning of each line are ***not*** part of the file, but only serve in the discussion that follows.

13.3.3.1 General Points

The algorithm configuration file is in Perl syntax. It is, in fact, a compilable Perl source file. As such, all the syntax rules that apply to Perl apply here as well. Although this Perl "code" is basically a list of parameter (or Perl variable) definitions, it does open up the possibility to add complex Perl code to this file. This will not, however, be discussed here.

Also note that typically, the order of parameters is not important. Thus, the $algorithm_name and $algorithm_version parameters can be set at the bottom of the file although, for the sake of maintainers, this may not be the wisest choice.

13.3.3.2 Line By Line Dissection

Line Numbers	Discussion
1-3	In lines 1-3, the algorithm name, version, and the name of the executable to run are set. Note that S4PM assumes that the executable has the correct permissions to be executed by the S4PM user.
4	The processing period is set to 300 seconds. This means that the algorithm will be processing 300 seconds of input. Typically, this means that the output corresponds to the same 300-second time span, although this doesn't have to be the case.
5-6	The $pre_processing_offset and $post_processing_offset are both set to 0. These parameters could have instead been left out of the file altogether and have the same effect. Because both are zero, this means that the processing period (which is 300 seconds) is aligned to the beginning of the timer period represented in the trigger input data. In other words, the processing period is contemporaneous with the trigger input data.
7	The $metadata_from_metfile being set to zero means that the metadata are read from the HDF file rather than from the accompanying metadata file. As with lines 5-6, this line could have been left out since the default for this parameter is zero.
8	The $apply_leapsec_correction being set to zero means that no such correction will be done; this is not an AIRS algorithm. The line could have been left out altogether.
9	The $pcf_path is set to the relative full pathname of the Process Control File (PCF) template for this algorithm. It will be from this template that the runtime PCF will be generated. The advantage of a relative path rather than an absolute one is that this configuration file is portable to any S4PM string whereas an absolute path may need to be changed when changing strings.
10	The $product_coverage is set to 300 seconds. This means that *all* products from this algorithm are assumed to be 300 seconds long.
14	Since $make_ph is set to non-zero (1 in this case), a Production History (PH) tar file will be generated when this algorithm completes successfully. Whether or not the PH get exported is dependent upon the $export_ph parameter in the Stringmaker string configuration file. If PH files are not exported, they will remain on disk in the PH disk pool until they are deleted by a job running in the Repeat Daily station after four days (4 is hardwired in the Stringmaker derived configuration file).
16	Since $run_easy is set to zero, the Run Easy algorithm wrapper will not be invoked.
18-59	Lines 18-59 are where the input files for this algorithm are described.
19-28	This section describes the data type MOD03, version 005. Note that since the need is set to 'TRIG', this data type is designated the trigger data type. Thus, it is the arrival of a file of MOD03 version 005 that triggers S4PM into action by setting up a job to determine what other data are needed by a run using this file as input and then to begin looking for those data. The coverage attribute is 300 seconds, equal to the processing period set

Line Numbers	Discussion
	earlier. The boundary is set to START_OF_DAY. S4PM will thus assume, when determining what other data are needed for this algorithm, that MOD03 aligned to the beginning of the data. Since MOD03 are only 300 seconds long, a boundary of START_OF_HOUR would have achieved equal results.
29-38	This section describes MOD01 version 005. Since the need is set to 'REQ', we know that this input is required. S4PM will not allow this job to run unless this data is available. We also note that the currency is set to 'CURR'. This means that the MOD01 is aligned exactly with the MOD03; what is "current" is dictated by the trigger input.
39-58	These sections describe two optional inputs. They are both MOD01, version 005. The distinction between these MOD01 files and the one described in lines 29-38 is that (1) they are for the previous and following files, and (2) both are optional. The currency setting of 'PREV1' means that the MOD01 is previous by one step to the current and "current" is defined by the time of the trigger input. So for example, if the trigger MOD03 was for 10:00-10:05, a 'PREV1' means times 09:55-10:00. If the need had been set to 'PREV2', it would have meant times 09:50-09:55, 'PREV3' would have meant 09:45-09:50, etc. Likewise, the second MOD01 has a currency of 'FOLL1' meaning a MOD01 from the time period immediately after the current. If current was 10:00-10:05, then 'FOLL1' means 10:05-10:10. Since the need attribute for both files is set to 'OPT1', we know these files are optional. Use them if they are available. How long does S4PM wait before processing without them? That is set by the timer attribute which we see is 7200 seconds for both. S4PM will wait for up to two hours for these optional inputs to arrive before giving up on them and running the algorithm without. Note that each of these two inputs has a different LUN (see the lun attribute). It is only because of this that we can ascribe different rules for each such as the currency.
61-85	These lines describe the outputs from this algorithm. Note that all *possible* outputs need to be listed here even if they are not always produced.
113-125	In these lines the input and output uses are set. All input and output data types need to be listed in these hashes. To determine the number of input uses for each data type, you need to ask 'What is the maximum number of times a data file of this data type will be accessed (read) by this algorithm?' NOTE: You are not asking how many times a single run of this algorithm will access the data (since the answer to this is almost always one), but how many times will *runs* of this algorithm access the data. For example, for a two-hour input file, you may have an algorithm that processes only 15 minutes of it at a time. Thus, to process the entire tow-hour file, the algorithm will have to run 8 times. Thus, the number of uses

Line Numbers	Discussion
	for this input is 8 since data needs to be read by runs of this algorithm 8 times before it has "used" it up. Do not concern yourself with what other algorithms may read or access the same input. S4PM will tally the total number of uses on any data type for you automatically. For outputs, the number of uses is almost always one (the export of a data file is considered a use of it).
127-130	In these lines, the @stats_datatypes and $stats_index_datatype parameters are set. In the former, you might notice that not all output data types are listed, namely the MOD021QA. This is because this QA output is not considered "important" and therefore, no performance statistics need be generated on it. The $stats_index_datatype is set to MOD021KM. This is because only the MOD021KM happens to be made in all runs. It turns out that the MOD02HKM and MOD02QKM are not always produced. So, they would have been a poor choice for the index data type.

Table 13-1. A line-by-line discussion of the algorithm configuration file shown in Appendix A.

13.4 Process Control Files

This section discusses Process Control Files or PCFs. These files are part of the algorithm package and have been mentioned earlier.

13.4.1 The Process Control File

S4PM generates a runtime Process Control File (PCF) for each run of every algorithm. The PCF is an ASCII text file that maps physical files and directories to logical unit numbers (LUNs). S4PM assigns input and output file names and directories for each run of an algorithm. The only way to have an algorithm access these dynamically generated names in a consistent way is to via the PCF. With a PCF, the algorithm accesses the LUN and uses that LUN to determine the current value of the file name and directory location. In addition to mapping files to LUNs, the PCF can also map parameters to LUNs. In the ECS world, these so-called user-defined parameters can be used to pass various values (numeric, character, etc.) to the running algorithm; they function very much like command line arguments.

The PCF format is based on that used with the ECS Toolkit, but simplified greatly.

For algorithms that do not use the ECS Toolkit (and the API for handling PCFs), S4PM provides tools to shield the algorithm from having to deal with them. Bear in mind, however, that PCFs are still used in the background by S4PM.

13.4.2 The Process Control File Template

As mentioned above, S4PM generates a new PCF for each algorithm run. What changes in the runtime PCF from run to run are the specific file names. In addition, because some input files may be optional, PCFs may also differ from run to run as far as what files they contain. For example, in one run, optional data may be left out because it did not arrive soon enough.

The PCF template is the template that S4PM uses to generate dynamic runtime PCFs. The PCF template is part of the algorithm package. Since S4PM uses the PCF template to generate a runtime PCF, the PCF template needs to contain *all possible* input files and *all possible* output files, both optional and required. In any one algorithm run, only those inputs actually found will end up in the runtime PCF. If an algorithm has no optional inputs or outputs, then the PCF template will look like the runtime PCF.

The file name for a PCF template in the algorithm package is unrestricted. The algorithm configuration file $pcf_path must be set to the relative or absolute pathname of the PCF template (see Section 10.1.1.8).

Appendix B contains an example PCF template corresponding to the algorithm configuration file in Appendix A.

Several points can be illustrated by the PCF template in Appendix B:

1. All sections of the PCF must be present, even if they contain no entries. These required sections are:
    a. PRODUCT INPUT FILES
    b. PRODUCT OUTPUT FILES
    c. SUPPORT INPUT FILES
    d. SUPPORT OUTPUT FILES
    e. USER DEFINED RUNTIME PARAMETERS
    f. INTERMEDIATE INPUT
    g. INTERMEDIATE OUTPUT
    h. TEMPORARY I/O
    i. END
2. Syntax must be followed. Section names must be in all uppercase and lines containing section names must begin with a ? character in the first column. See lines 2, 29, 39, 42, 48, 70, 73, and 76.
3. Section names must then be followed by a line with a ! in the first column (comment lines may be inserted in between) followed by a default directory name. In S4PM, the output directory must be set to ./ meaning the current directory. See lines 30, 43, and 77.
4. There can be ***NO*** blank lines in a PCF or PCF template. A line must either be a PCF entry or a comment line beginning with the # character (even if there is no comment afterward). See lines 1, 25, and 28 for example.

5. For static input files (*e.g.* lookup tables), the PCF entry in the template must be the actual file name and directory. Directories should be relative as shown in lines 5 through 7. Remember that static files are not seen by S4PM.
6. For dynamic input, the only important part of the entry is the LUN. The file name that you place in the PCF (and there needs to be something there) is arbitrary and only serves the human reader of the document. See lines 9 through 13.
7. Metadata configuration files (MCFs), if your algorithm uses them, are like static input files. The file names and directory locations need to be real. See lines 18 through 23.
8. Static parameters that need to be passed to the algorithm can be set in the USER DEFINED RUNTIME PARAMETERS section as in lines 49 through 68. Remember though, these are static. They will not change from one run to the next. (The Modify PCF Parameters Tool does allow operators to change these parameters, but it is not meant to be applied on a run-by-run basis.)
9. The last line must be END. See line 79.

For more information on PCF syntax, refer to the ECS Toolkit User's Guide.

13.5 Preparing an Algorithm Package for S4PM

An algorithm package in S4PM consists of the following components:

Component	Description	Required?
Executable binaries and scripts	Algorithm code can be any combination of compiled or script binaries. Permissions must be properly set so that they can be executed on the host machine by the S4PM user. One binary (compiled or script) must be designated as the main program. It will be this program that S4PM will execute. The main program may then execute other compiled or script binaries as needed. The main program, however, must assume that these other binaries are in the current working directory.	Required
Algorithm Configuration File	There must be at lest one algorithm configuration file as described in Section 10.	Required
PCF Template	There must be exactly one Process Control File template.	Required
Static Input Files	Static inputs (lookup tables, flat file databases, etc.) are packaged with the algorithm. Static input are not described by the algorithm configuration file, but their locations must be hardwired in the PCF template. Unlike with dynamic input, static input files are not “seen” by S4PM. It may be convenient to place static input in a subdirectory (say, named ‘static’).	Optional
Metadata Configuration Files	Metadata configuration files or MCFs are needed by algorithms that use the ECS Toolkit. If included, the location of MCFs needs to be hardwired in the PCF template (as with other static input).	Optional
Other Files	Algorithm packages may include any other files to support policies at your site. For example, README files, output from software builds, or other documentation.	Optional

Table 13-2. Required and optional components of the algorithm package in S4PM.

Typically, the components comprising an algorithm package are placed into one or more tar or zip files.

13.6 Installing Algorithm Packages

Algorithm package installation into S4PM is a two-step process. The first step is to simply install the algorithm package into the disk location where S4PM is configured to look. The second step is to configure S4PM to incorporate the algorithm into its processing.

13.6.1 Installation

Algorithm packages must be installed under the algorithm root directory defined by the $algorithm_root parameter in the Stringmaker string configuration file as described in

Section 9.7 or in the default location: $s4pm_root/$data_source/pge where $s4pm_root and $data_source are parameters also set in the Stringmaker string configuration file.

Below the algorithm root directory, S4PM assumes there is a subdirectory for each algorithm with the name of the algorithm (matching the $algorithm_name parameter in the algorithm configuration file). Then below this directory, S4PM assumes that there is a subdirectory named for the algorithm version (matching the $algorithm_version parameter in the algorithm configuration file).

For example, algorithm AiL1A_AIRS, version 2.3.4 is assumed to reside in:

$algorithm_root/AiL1A_AIRS/2.3.4/

→ **NOTE:** S4PM does not "see" an algorithm until it has been configured to do so. Therefore, you can safely install (place on disk) any number of algorithms or algorithm versions without affecting current processing.

13.6.2 Configuring S4PM For An Algorithm

See Sections 4.3.10 and 4.3.11 for how to run Stringmaker.

Appendix A. Sample Stringmaker Algorithm Configuration File

The following lines form an example Stringmaker algorithm configuration file, in this case, for MoPGE02. Note that the line numbers shown in the first column are ***not*** part of the configuration file, but are used only to discuss this file in Section 13.

```
 1    $algorithm_name = 'MoPGE02';
 2    $algorithm_version = '5.0.6';
 3    $algorithm_exec = 'PGE02.csh';
 4    $processing_period = 300;
 5    $pre_processing_offset = 0;
 6    $post_processing_offset = 0;
 7    $metadata_from_metfile = 0;
 8    $apply_leapsec_correction = 0;
 9    $pcf_path = '../MoPGE02/5.0.6/GDAAC.PGE02.pcf.tpl';
10    $product_coverage = 300;
11
12
13    # CHANGE THE SETTING BELOW IF YOU WANT PH FILES MADE
14    $make_ph = 1;
15
16    $run_easy = 0;
17
18    %inputs = (
19        'input1' => {
20            'data_type'    => 'MOD03',
21            'data_version' => '005',
22            'need'         => 'TRIG',
23            'lun'          => '600000',
24            'timer'        => 7200,
25            'currency'     => 'CURR',
26            'coverage'     => 300,
27            'boundary'     => 'START_OF_DAY',
28        },
29        'input2' => {
30            'data_type'    => 'MOD01',
31            'data_version' => '005',
32            'need'         => 'REQ',
33            'lun'          => '500000',
34            'timer'        => 7200,
35            'currency'     => 'CURR',
36            'coverage'     => 300,
37            'boundary'     => 'START_OF_DAY',
38        },
39        'input3' => {
40            'data_type'    => 'MOD01',
41            'data_version' => '005',
42            'need'         => 'OPT1',
43            'lun'          => '500001',
44            'timer'        => 7200,
45            'currency'     => 'PREV1',
46            'coverage'     => 300,
47            'boundary'     => 'START_OF_DAY',
48        },
49        'input4' => {
50            'data_type'    => 'MOD01',
51            'data_version' => '005',
```

```
            'need'          => 'OPT1',
            'lun'           => '500002',
            'timer'         => 7200,
            'currency'      => 'FOLL1',
            'coverage'      => 300,
            'boundary'      => 'START_OF_DAY',
        },
    );

    %outputs = (
        'output1' => {
            'data_type'    => 'MOD021KM',
            'data_version' => '005',
            'lun'          => '700002',
            'currency'     => 'CURR',
            'coverage'     => 300,
            'boundary'     => 'START_OF_DAY',
        },
        'output2' => {
            'data_type'    => 'MOD02QKM',
            'data_version' => '005',
            'lun'          => '700000',
            'currency'     => 'CURR',
            'coverage'     => 300,
            'boundary'     => 'START_OF_DAY',
        },
        'output3' => {
            'data_type'    => 'MOD02OBC',
            'data_version' => '005',
            'lun'          => '700010',
            'currency'     => 'CURR',
            'coverage'     => 300,
            'boundary'     => 'START_OF_DAY',
       },

        'output4' => {
            'data_type'    => 'MOD021QA',
            'data_version' => '005',
            'lun'          => '700100',
            'currency'     => 'CURR',
            'coverage'     => 300,
            'boundary'     => 'START_OF_DAY',
        },
        'output5' => {
            'data_type'    => 'MOD02HKM',
            'data_version' => '005',
            'lun'          => '700001',
            'currency'     => 'CURR',
            'coverage'     => 300,
            'boundary'     => 'START_OF_DAY',
        },
        'output6' => {
            'data_type'    => 'Browse',
            'data_version' => '001',
            'lun'          => '99201',
            'currency'     => 'CURR',
            'coverage'     => 300,
            'boundary'     => 'START_OF_DAY',
       },
    );

    %input_uses = (
        'MOD01' => 3,
        'MOD03' => 1,
```

```
116     );
117
118     %output_uses = (
119         'MOD02HKM' => 1,
120         'MOD02QKM' => 1,
121         'MOD021KM' => 1,
122         'Browse' => 1,
123         'MOD021QA' => 1,
124         'MOD02OBC' => 1,
125     );
126
127     @stats_datatypes = ('MOD021KM','MOD02QKM',
128         'MOD02OBC', 'MOD02HKM', 'Browse', );
129
130     $stats_index_datatype = 'MOD021KM';
131
132     1;
```

Appendix B. Sample Process Control File

The following is the Process Control File for the same algorithm described in the algorithm configuration file of Appendix A. Note that the line numbers shown in the first column are ***not*** part of the configuration file, but are used only to discuss this file in Section 13.

```
  1    #
  2    ?   PRODUCT INPUT FILES
  3    !  ~/runtime
  4    # Static Input Lookup Tables
  5    700050|MOD02_Reflective_LUTs.hdf|../MoPGE02/5.0.6/pge/static||||1
  6    700060|MOD02_Emissive_LUTs.hdf|../MoPGE02/5.0.6/pge/static||||1
  7    700070|MOD02_QA_LUTs.hdf|../MoPGE02/5.0.6/pge/static||||1
  8    # Geolocation
  9    600000|MOD03.A2003186.0125.004.2003186082217.hdf|||||1
 10   # L1A input files below (Also see 70020,700223,700222,201001)
 11   500000|MOD01.A2003186.0120.004.2003186081241.hdf|||||1
 12   500001|MOD01.A2003186.0125.004.2003186081527.hdf|||||1
 13   500002|MOD01.A2003186.0130.004.2003186081714.hdf|||||1
 14   #----------------------------------------------------------------
-------
 15   10252|GetAttr.temp|./||||1
 16   10254|MCFWrite.temp|./||||1
 17   # MCFs
 18   700250|MOD02QKM#005.MCF|../MoPGE02/5.0.6/pge/static||||1
 19   700251|MOD02HKM#005.MCF|../MoPGE02/5.0.6/pge/static||||1
 20   700252|MOD021KM#005.MCF|../MoPGE02/5.0.6/pge/static||||1
 21   700253|MOD02OBC#005.MCF|../MoPGE02/5.0.6/pge/static||||1
 22   700350|MOD021QA#005.MCF|../MoPGE02/5.0.6/pge/static||||1
 23   10250|Browse#001.MCF|../MoPGE02/5.0.6/pge/static||||1
 24   # ----------------------------------------------------------------
-------
 25   #
 26   10501|INSERT_EPHEMERIS_FILES_HERE|||||1
 27   10502|INSERT_ATTITUDE_FILES_HERE|||||1
 28   #
 29   ?   PRODUCT OUTPUT FILES
 30   ! ./
 31   700000|MOD02QKM.hdf|||||1
 32   700001|MOD02HKM.hdf|||||1
 33   700002|MOD021KM.hdf|||||1
 34   700010|MOD02OBC.hdf|||||1
 35   700100|MOD021QA.hdf|||||1
 36   # MOD_PR02BR (Browse) Output File:
 37   99201|Browse.Terra.Af5.3_6||| ||1
 38   #
 39   ?   SUPPORT INPUT FILES
 40   !  ~/runtime
 41   #
 42   ?   SUPPORT OUTPUT FILES
 43   ! ./
 44   10100|LogStatus|||||1
 45   10101|LogReport|||||1
 46   10102|LogUser|||||1
 47   #
 48   ?   USER DEFINED RUNTIME PARAMETERS
 49   700200|ECS METADATA|700100:1
 50   700201|ECS METADATA|700101:1
 51  700202|ECS METADATA:700101:1
```

```
52   800500|PGE02 Version|5.0.6
53   800550|Processing Environment|IRIX64
54   800510|Satellite; AM1M=Terra, PM1M=Aqua|AM1M
55   800600|ReprocessingPlanned|further update is anticipated
56   800605|ReprocessingActual|processed once
57   800610|MCSTLUTVersion|5.0.6.4_Terra
58   800615|Write_Night_Mode_HiRes_Data|0
59   800620|ProcessingCenter|GSFC
60   # MOD_PR02BR (Browse) parameters:
61   # Number of Input bands
62   99401|band1|1
63   99402|band2|4
64   99403|band3|3
65   99406|browse_product_shutdown|0
66   # ShortName & VersionID are used in MCF
67   99505|SHORTNAME|"DFLAXDUMMY"
68   99506|VERSIONID|"V01"
69   #
70   ?   INTERMEDIATE INPUT
71   !  ~/runtime
72   #
73   ?   INTERMEDIATE OUTPUT
74   ~/runtime
75   #
76   ?   TEMPORARY I/O
77   ! ./
78   #
79   ?   END
```