XC™ Series Configurator User Guide

(CLE 6.0.UP07)

S-2560

Contents

1 About the XC™ Series Configurator User Guide

The XC™ Series Configurator User Guide (S-2560) describes the configurator tool within the context of the Cray configuration management framework (CMF) and provides procedures and examples for using that tool both interactively and with configuration worksheets.

Release CLE 6.0.UP07

This publication of XC™ Series Configurator User Guide supports Cray software release CLE 6.0.UP07, released on 12 JUL 2018.

New in this release

- There are no major revisions to this publication for the CLE 6.0.UP07 release.

Audience and Scope

This publication is intended for system installers, administrators, and anyone who configures software services on a Cray XC™ Series system. Use of the term user throughout refers to the intended audience, not to end users of the system.

Command Prompt Conventions

Host name and account in command prompts

The host name in a command prompt indicates where the command must be run. The account that must run the command is also indicated in the prompt.

- The `root` or super-user account always has the # character at the end of the prompt.
- Any non-`root` account is indicated with *`account@hostname`*>. A user account that is neither `root` nor `crayadm` is referred to as `user`.

`smw#`	Run the command on the SMW as `root`.
`cmc#`	Run the command on the CMC as `root`.
`sdb#`	Run the command on the SDB node as `root`.
`crayadm@boot>`	Run the command on the boot node as the `crayadm` user.
`user@login>`	Run the command on any login node as any non-`root` user.
`hostname``#`	Run the command on the specified system as `root`.

`user@hostname>`	Run the command on the specified system as any non-`root` user.
`smw1#` `smw2#`	For a system configured with the SMW failover feature there are two SMWs—one in an active role and the other in a passive role. The SMW that is active at the start of a procedure is smw1. The SMW that is passive is smw2.
`smwactive#` `smwpassive#`	In some scenarios, the active SMW is smw1 at the start of a procedure—then the procedure requires a failover to the other SMW. In this case, the documentation will continue to refer to the formerly active SMW as smw1, even though smw2 is now the active SMW. If further clarification is needed in a procedure, the active SMW will be called smwactive and the passive SMW will be called smwpassive.

Command prompt inside chroot

If the `chroot` command is used, the prompt changes to indicate that it is inside a chroot environment on the system.

```
smw# chroot /path/to/chroot
chroot-smw#
```

Directory path in command prompt

Example prompts do not include the directory path, because long paths can reduce the clarity of examples. Most of the time, the command can be executed from any directory. When it matters which directory the command is invoked within, the `cd` command is used to change into the directory, and the directory is referenced with a period (`.`) to indicate the current directory.

For example, here are actual prompts as they appear on the system:

```
smw:~ # cd /etc
smw:/etc# cd /var/tmp
smw:/var/tmp# ls ./file
smw:/var/tmp# su - crayadm
crayadm@smw:~> cd /usr/bin
crayadm@smw:/usr/bin> ./command
```

And here are the same prompts as they appear in this publication:

```
smw# cd /etc
smw# cd /var/tmp
smw# ls ./file
smw# su - crayadm
crayadm@smw> cd /usr/bin
crayadm@smw> ./command
```

Typographic Conventions

`Monospace`	Indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, and other software constructs.
`Monospaced Bold`	Indicates commands that must be entered on a command line or in response to an interactive prompt.

Oblique or *Italics*	Indicates user-supplied values in commands or syntax definitions.
Proportional Bold	Indicates a **GUI Window**, **GUI element**, cascading menu (e.g., **Ctrl**→**Alt**→**Delete**), or key strokes (e.g., press **Enter**).
\ (backslash)	At the end of a command line, indicates the Linux® shell line continuation character (lines joined by a backslash are parsed as a single line).

Trademarks

2 Introduction to the Cray Configurator

To configure Cray XC systems (CLE 6.0 and later) and manage configuration content, system installers and administrators use the Cray configuration management framework (CMF). The CMF comprises configuration data, the tools to manage and distribute that data, and software to apply the configuration data to the running image at boot time. The `cfgset` command and the configurator that it invokes are the primary tools that Cray provides for managing configuration data. This user guide describes all of the `cfgset` subcommands and many of the options needed to manage configuration data; however it focuses on the subset of subcommands and options that invoke the configurator tool. Use of the term user throughout refers to system installers and administrators, not to end users of the system.

The Configurator and the CMF Components it Touches

Within the CMF, configuration begins with the installation of configuration service packages on the system's management node. Then `cfgset` is used to pull configuration content from those service packages to create a central repository called a configuration set or config set, which is where the Cray Linux Environment (CLE) stores all configuration information necessary to operate the system. The configurator, invoked through the `cfgset` command, finds configuration templates from each service package that match the config set type and copies them into the config set. The configurator then gathers site-specific configuration data interactively or through bulk upload of data entered in configuration worksheets (optional), merges any new data with the data in the config set that was pulled from the service packages, and writes it out to config set templates and configuration worksheets.

To complete the picture, configuration continues (without the help of the configurator) through the actions of other CMF components. The IMPS Distribution Service (IDS) makes config sets, resident on the management node, available to all nodes in the system as read-only content. At each node, config set data is consumed by Ansible plays, another component of the CMF, which act upon that data during the booting phase to enable each node to dynamically self-configure.

2.1 About Configuration Service Packages

Configuration content (data and software) is installed as configuration service packages on the management node of Cray XC systems (in `/opt/cray/imps_config/<service package>/default/configurator` by default). Each service package delivers configuration content for one or more system services. Cray-supplied configuration service packages may be augmented by site-supplied service packages as well, if sites wish to create their own. In both cases, the contents of each service package reside in the following subdirectories:

`ansible`	Ansible play content.
`callbacks`	Pre- and post-configuration scripts.
`dist`	Other content, such as static files required for the configuration of a service.
`template`	Configuration templates that define the configuration settings for a service and provide some default values. These templates are never modified by administrators or other users.

`rules` Validation rules scripts are located here.

Configuration service packages are installed for system updates and upgrades as well as for initial installation.

Configuration Templates

Configuration templates define the parameters (configuration data) that are required by the system to dynamically self-configure. Each template defines a single service, which is a logical grouping of data needed to configure that service (networking, time, user settings, etc.). The data for configuring a service is composed of settings, which act as logical groupings of configuration data within a service. Configuration templates may include pre-populated data (default values) provided by the service package as a convenience or to specify values that are required for the service to function. Configuration templates in a configuration service package are never modified by sites. When configuration template content is pulled into a config set, the configuration templates created within the config set are modified by the site and become the repository for all the data needed to define the services.

Templates are implemented as YAML files and adhere to a specific, versioned schema. The configuration template schema defines general service-level attributes as well as the validation, documentation, and format of configuration data for each service. The template naming convention is `<site>_<service>_config.yaml`. Cray-provided configuration templates use the prefix `cray_` as the `<site>` portion of the file name. For example, the Cray-provided template for the networking service is `cray_net_config.yaml`.

Callback Scripts

When config sets are created or updated, scripts found in the service package `callbacks` subdirectory are executed prior to and immediately after a configurator session. These scripts act on the data in the configuration templates and create content necessary for system configuration, which they place into the `files` subdirectory of a config set when it is created or updated. All files in the service package callbacks location are executed if they are marked as executable in the file system.

Ansible and Dist Content

The configuration of a service on the system may require certain static files or Ansible content to be included in the config set for use in the rest of the system. The `ansible` and `dist` subdirectories act as repositories for that content. Each time a config set is updated, the content from these subdirectories overwrites content in the config set that originated from the service package.

Validation Rules

Validation rules are scripts that are run by the `cfgset validate` command. These Cray-provided scripts are used to provide validation beyond the basic checking done by the configurator. Examples include checking the correctness of settings within services, between services, and between global and non-global config sets.

2.2 About Config Sets

Users invoke the `cfgset` command to take configuration content delivered in service packages and combine it with site-specific configuration content gathered either interactively or through bulk import. The results are used by `cfgset` to create a config set, which is a central repository that stores all configuration information necessary to operate the system. Config sets reside on the management node (e.g., the SMW)

in `/var/opt/cray/imps/config/sets` by default. The contents of each config set reside in the following subdirectories:

`ansible` Local site-provided Ansible play content can be placed here for distribution with the config set. When the config set is created, `cfgset` copies Ansible content from service packages to this location. Whenever the config set is updated, `cfgset` copies Ansible content from service packages again, overwriting the previous service-package Ansible content and leaving the site-provided content unchanged.

`changelog` YAML change logs from previous sessions with the configurator.

`config` Configuration templates containing configuration information. When the config set is created, the configurator copies service package templates to this location. Users can modify the content of these templates using `cfgset` to invoke the configurator. Whenever the config set is updated, the configurator merges service package templates with the templates in this location.

`dist` Other site-provided content, such as static files required for the configuration of a service, can be placed here for distribution with the config set. When the config set is created, `cfgset` copies dist content from service packages to this location. Whenever the config set is updated, `cfgset` copies dist content from service packages again, overwriting the previous service-package dist content and leaving the site-provided content unchanged.

`files` Files necessary for system configuration that are distributed with the config set. They can be placed here by:

- the `cfgset` command, which runs configuration callback scripts to generate some configuration files (e.g., `/etc/hosts`)
- the Simple Sync service
- local site administrators

`worksheets` Configuration worksheets generated by the configurator using data stored in the configuration templates in the `config` subdirectory of the config set. Administrators copy these worksheets to a location outside the config set, edit them with site-specific configuration data, and then import them to create a new config set or update an existing one.

Config Set Types

All config sets have a type associated with them that is specified upon creation. XC systems require both a `global` config set type and a `cle` config set type. After a config set of a given type is created, its type cannot be changed. A user may create multiple config sets to support partitioned systems or alternate configurations. Typically a config set of type `cle` is created for each partition to store partition- and CLE-specific content, and another config set of type `global` is created to store configuration data that pertains to the management node domain as well as configuration data that can be easily shared among `cle` config sets. Config sets can be portable between partitions or to other systems if their partition-specific information is modified accordingly.

Configuration Service Inheritance

When a config set is created or updated, only service package templates that match the type of the config set can be included in the config set. Cray provides several service package templates that match both types and can be included in both `cle` and `global` config sets. In such cases, the user can choose which template will be used to configure the service in question. When a `cle` config set is created, and a service that has a template of both types is ready for configuration, the configurator will inject an initial question for the user to choose between configuring the service (i.e., using the `cle` version of the template) or letting the service inherit configuration values from the `global` config set (i.e., inheriting values from the `global` version of the template). Configuration

worksheets for such services also provide that choice by including an inherit field, which can be set to true or false. If the user sets it to true, the configuration data from the global config set version of the service will be used. When the Cray-provided `cray-ansible` service (part of the Cray Configuration Management Framework) is run at boot time or at the system administrator's discretion, it uses the value of the inherit field to determine which configuration template data (`global` or `cle`) to use.

Inheritance is useful for systems with multiple partitions where a subset of partitions need custom configuration of a service, but another subset of partitions can all share the same global configuration.

2.3 About Configuration Worksheets

Configuration worksheets are a means for users or automated processes to import configuration data directly into a config set instead of using the configurator user interface. In some situations, large amounts of information may be needed to configure a specific service or setting in the configuration data (for example, network interfaces), and gathering this information through an interactive user interface may be too time consuming during a fresh install. Furthermore, there may be a benefit to gathering configuration data offline prior to system installation and storing it in a worksheet for use later when the system is ready.

Worksheets are generated each time a config set is created and updated each time a config set is updated. They are not delivered with the configuration service package itself, but generated from configuration templates in service packages and placed into the config set. Like configuration templates, worksheets are valid YAML files and therefore are both human- and machine-readable. Many modern programming languages provide interfaces to YAML files, thereby making it possible to use configuration worksheets to create and update config sets through automated processes.

See *Create a Config Set from Configuration Worksheets* on page 19 or *Update a Config Set from Configuration Worksheets* on page 24 for instructions on how to create and update config sets using configuration worksheets. Also, see *Common Tasks When Using Configuration Worksheets for Bulk Import* on page 76 for instructions on how to prepare configuration worksheets for use.

2.4 About Variable Names in the Configurator and Configuration Worksheets

In the configurator and configuration worksheets, variable names can be quite long because they are composed of a data structure hierarchy. Each variable name begins with the name of the service to which it belongs. The next part of each name is always 'settings' to indicate that what follows is a service setting, one of the available settings for that service. After 'settings' comes the name of the setting, which could be a simple data type (string, boolean, integer, etc.) or a more complex data type (list, multival, etc.). The next part after the name of the setting is always 'data' to indicate that what follows is one of the fields of that setting. For a full description of data types, see *XC™ Series Configurator User Guide* (S-2560).

For example, here is the variable for the IP address of the high-speed network (HSN), one of several networks.

```
cray_net.settings.networks.data.hsn.ipv4_network
```

This variable belongs to the `cray_net` service and the `networks` setting of that service. The networks setting is of type multival, which means it can have multiple entries, and each entry can have multiple fields to set. This variable targets the `ipv4_network` field of the `hsn` network entry.

This example shows the variable for the IP address of the HSN SDB node alias interface (one of several interfaces) of the SDB node (one of several hosts).

```
cray_net.settings.hosts.data.sdbnode.interfaces.hsn_sdb_alias.ipv4_address
```

This variable belongs to the `cray_net` service and the `hosts` setting of that service. The hosts setting is of type multival, and this variable belongs to the `sdbnode` host entry. The sdb_node host has a field `interfaces`, which is also of type multival. This variable targets the `ipv4_address` field of the `hsn_sdb_alias` interface entry.

3 Config Set and Configurator Operations

The `cfgset` command, invoked as root, can be used to perform all necessary operations on config sets: creating, viewing, updating, copying, comparing, searching, validating, and deleting.

CAUTION: It is possible for a user or process to modify config set content outside of the `cfgset` command. For Cray recommendations, see *Manually Edit Configuration Files* on page 34.

Config Set Operations

Here is a list of the `cfgset` subcommands that are used to perform all config set operations.

`create`	Create new config sets, clone existing config sets, and prepare configuration worksheets.
`diff`	Show changes between files in config sets.
`list`	List information for a config set. If no config set specified, list information for all config sets.
`push`	Copy contents of a config set to a remote node.
`remove`	Remove a config set when no longer needed. See *Remove a Config Set* on page 40.
`search`	Search configuration data in config sets for certain attributes and values.
`show`	Display an audit (or metadata) trail of actions taken on a config set.
`update`	Modify the attributes or contents of a config set or rename it.
`validate`	Check a config set for syntax, regular expressions, structure, etc.
`get`	Retrieve configuration data values of the given path using the command line interface.
`modify`	Change config set data using the command line interface.

In examples provided throughout, command line options to the `cfgset` command and its subcommands use the long form of each option (e.g., `--mode` instead of `-m`). To see the full set of command line options and their short and long forms, see the `cfgset` man page or the output of `cfgset` *`SUBCOMMAND`* `-h`.

Configurator Operations

Configurator operations are an important subset of config set operations. The configurator is responsible for all operations involving configuration templates and worksheets: copying, merging, updating, writing, and validating. It is also responsible for providing an interface that enables users to add or change configuration data interactively or through the import of configuration worksheets.

The configurator can be invoked only by using `cfgset` with one of the following subcommands. There is no way to invoke the configurator outside of the `cfgset` command.

`create` (except when the `--clone` option used)
`update`

```
search
validate
```

The configurator plays a major role in the *Config Set Create/Update Process* on page 13. The options selected for the `create` and `update` subcommands determine whether the configurator is run with or without user interaction (see *Use Mode to Choose How to Interact with the Configurator* on page 16) and which settings can be viewed and set by a user (see *Use Filters to Choose What to See with the Configurator* on page 17).

For details about configurator operations, see the following:

- *Create a Config Set* on page 18
- *Update a Config Set* on page 22
- *Search a Config Set* on page 27
- *Validate a Config Set and List Validation Rules* on page 38

3.1 Config Set Create/Update Process

Config sets are created and updated using the `cfgset` command with the `create` and `update` subcommands, respectively. Invoking `cfgset` with one of those subcommands initiates the following process, which defines how configuration content is discovered from service packages installed on the management node and used, along with site-supplied content, to create or update a config set.

1. `cfgset` searches for service packages in `/opt/cray/imps_config`.
2. `cfgset` copies to the config set (for `create`) or overwrites in the config set (for `update`) ansible and dist content from each service package. Note that it is only content from service packages that is overwritten; content placed in those directories manually is unchanged.

   NOTE: Manual changes to service package content in this directory will be overwritten!
3. `cfgset` runs pre-configuration callback scripts from each service package. Scripts act on the config set to create content necessary for system configuration, which they place into the `files` subdirectory of the config set.
4. `cfgset` invokes the configurator to do steps 4 through 6.

   Configurator finds configuration templates from each service package that match the config set type, and then copies them into the config set (for `create`) or merges them with the templates already in the config set (for `update`).
5. Configurator takes *one* of these actions to further modify config set template data, depending on the command-line options used:

interacts with user	Initiates an interactive session with the user and modifies config set template data based on the values supplied by the user. Occurs when `--mode interactive` option used or no mode option used, which defaults to `auto` mode.
does not interact with user	Does not initiate an interactive session and does no further modification to config set template data beyond the copy/merge of service package data already done in step 4.

	Occurs when `--mode prepare` option used. Note that although this action is associated with preparing worksheets, all three actions result in worksheets being written in step 6.
imports worksheets	Imports configuration worksheets and modifies config set template data based on the values in each service worksheet. Occurs when `--worksheet-path` *`FILEPATH`* option used.

6. Configurator writes configuration template data, configuration worksheets, and a changelog to the config set. Note that the configurator never modifies the configuration templates in service packages, which are found in `/opt/cray/imps_config/`*`SERVICE PACKAGE`* for each service package.
7. `cfgset` runs post-configuration callback scripts from each service package.
8. `cfgset` autosaves the config set to a time-stamped clone.

The following three figures illustrate how this eight-step process is used to create a CLE config set. They differ in how configuration data in a config set is further modified in step 5, corresponding to the three different actions: interacting with the user (modification through user interaction), not interacting with the user (no further modification), and importing worksheets (modification through bulk import of configuration worksheets). Black lines indicate `cfgset` actions, and red lines indicate actions taken by the configurator when invoked by `cfgset`.

This first figure shows how the configurator creates config set templates (in the `config` subdirectory) from service package templates in step 4, enables the user to enter new or modify existing configuration data in step 5, and then saves the new/modified data to the config set templates and worksheets in step 6.

Figure 1. Process to Create a Config Set Interactively

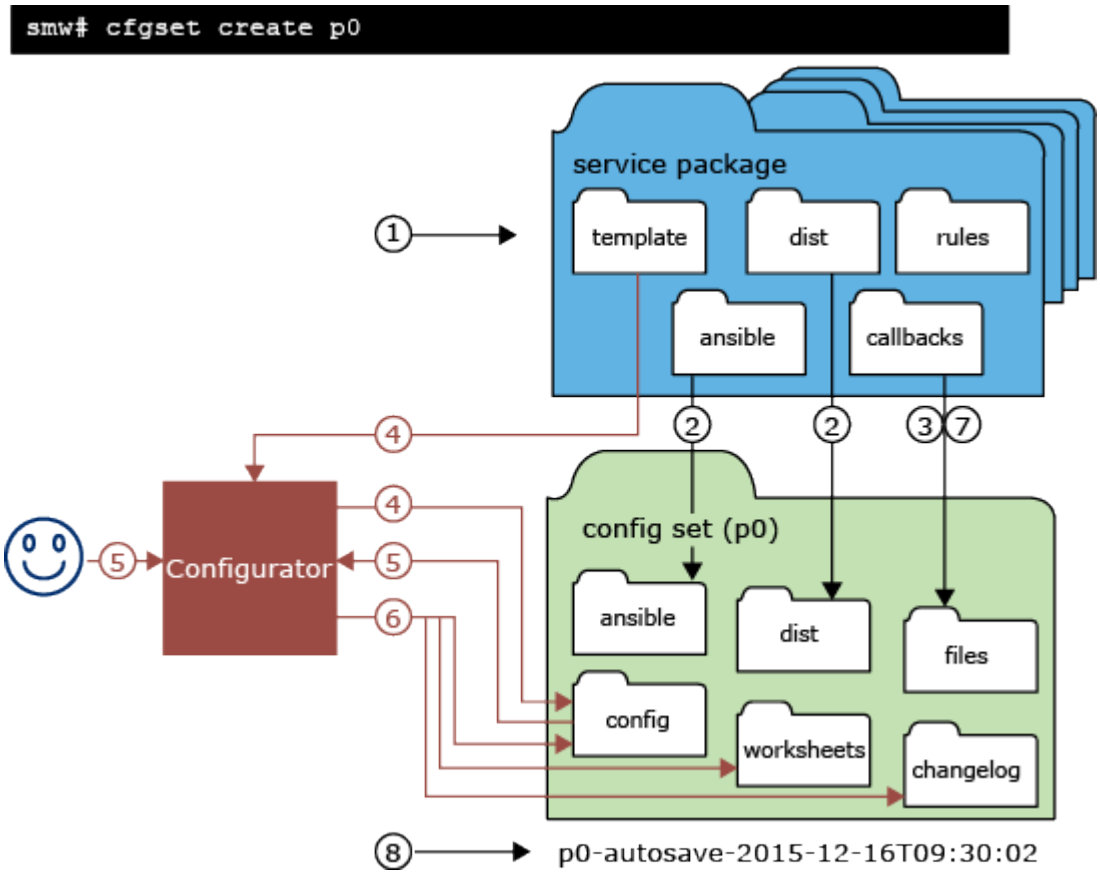
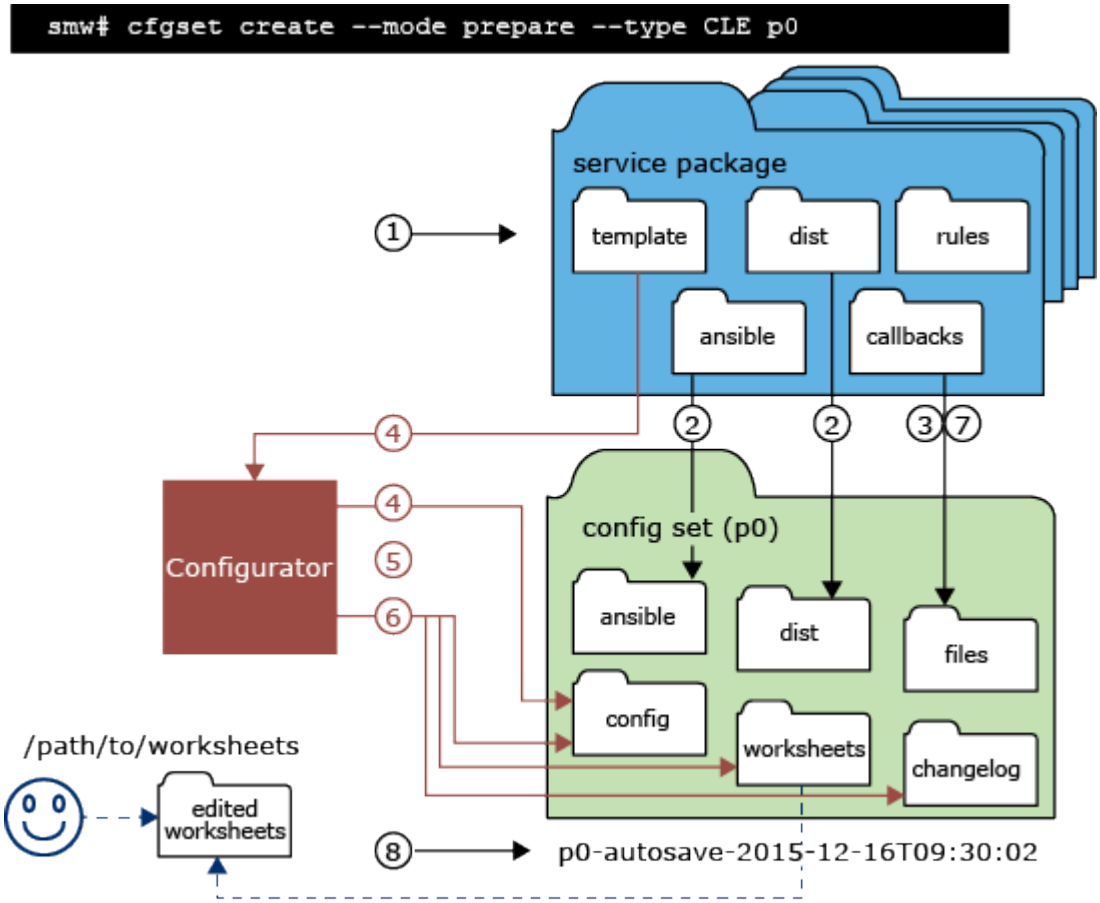
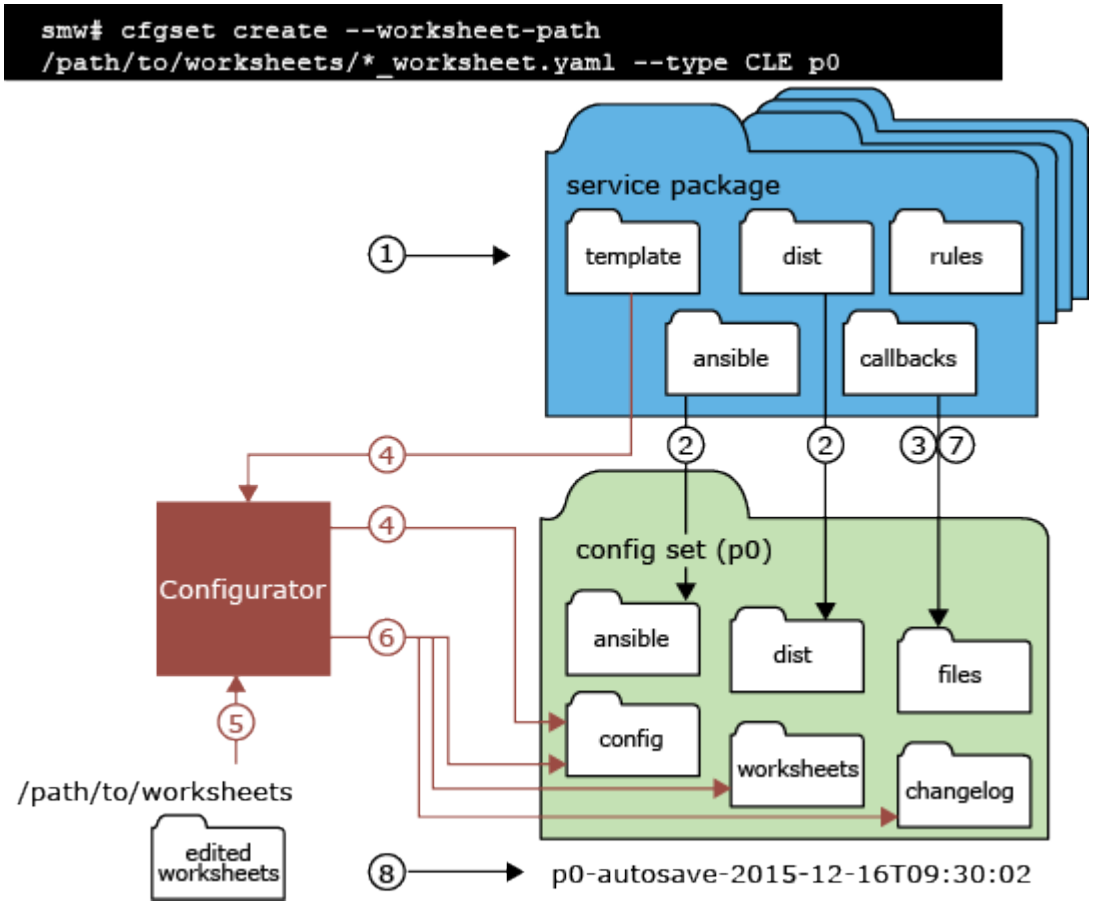


Figure 2. Process to Create a Config Set and Prepare Worksheets



The prepare-mode figure shows how the configurator creates config set templates from service package templates in step 4, does nothing to that configuration data in step 5, and then saves the data from step 4 to config set templates and worksheets in step 6. The blue dashed line indicates an action taken by the user after `cfgset` has completed the create/update process to prepare worksheets. The user (usually an installer or system administrator) copies the worksheets prepared by the configurator to a location outside the config set and edits them (or has other site staff edit them) with site-specific configuration values. It is these edited worksheets that are used when creating (or updating) a config set from worksheets (shown in worksheets figure).

Figure 3. Process to Create a Config Set from Worksheets



The worksheets figure shows how the configurator creates config set templates from service package templates in step 4, imports new or modified configuration data from worksheets in step 5, and then saves the new/modified data to the config set templates and worksheets in step 6.

3.2 Use Mode to Choose How to Interact with the Configurator

The `mode` option of the `cfgset` command determines how the configurator interacts with a user. Mode can be specified only with subcommands `create` and `update`.

`--mode` | `-m` Possible values: `auto` (default), `interactive`, `prepare`

In all modes, the configurator begins by copying/merging service package templates to config set templates (step 4 of the *Config Set Create/Update Process*). And in all modes, the configurator ends by writing templates, worksheets, and logs to the config set (step 6 of that process). What differentiates these three modes is how the configurator behaves in the middle (step 5 of the process): whether it initiates an interactive session with the user, what services and settings it presents, who ends the session, and whether changes are saved. That behavior is described for each mode here:

auto The configurator searches through all available configuration templates in the config set. If there are any configuration settings that meet state and level filtering criteria, the configurator initiates an interactive session with the user and presents those settings one at a time in a certain order (taking into account dependencies among services). When all of those settings have been presented to the user, the configurator automatically ends the interactive session and saves the config set. In this mode, if there are no settings that meet the specified filtering criteria, no interactive session is initiated.

interactive The configurator searches through templates as with auto mode, but in interactive mode, it presents a menu of all available services (or a menu of all available settings, when a service has been selected) that meet state and level filtering criteria. This mode enables the user to navigate through the services and settings to view and modify setting values as needed. The user chooses when to end the interactive session by exiting the configurator, and the user chooses whether to save any changes to the config set upon exit. In this mode, an interactive session is always initiated.

prepare The configurator does not initiate an interactive session with the user. The results of merging service package templates with config set templates are written to the config set templates and worksheets without any opportunity for modification by the user. This mode is typically used when a site wants to generate the most up-to-date templates or worksheets for editing to add site-specific configuration.

Note that when creating or updating a config set using configuration worksheets, the configurator ignores mode.

3.3 Use Filters to Choose What to See with the Configurator

Two `cfgset` command options act as filters to determine which settings are available to view and set or change. These options can be specified only with subcommands `create`, `update`, and `search`.

`--state|-S` Possible values: `unset` (default), `set`, `all`

`--level|-l` Possible values: `required`, `basic` (default), `advanced`

State The configurator keeps track of the configuration state of each service and each of its settings. A field will be marked as configured if any of the following conditions are met:

- The value for the field is set in the interactive configurator user interface. In this case, the prompt for the field is answered with anything other than the > response which skips configuration of the field.
- The value for the field is set using the `cfgset modify` command.
- The value for the field is set by importing a configurator worksheet that has an uncommented line specifying a value for the field.
- The configurator template in the config set is manually edited to set the `configured` field of the corresponding argspec to `true` (multival settings must also update the `unconfigured_keys` field).
- The installed template has the `configured` field of the corresponding argspec set to `true` by default.

unset Settings that have never been configured.

`set`	Settings that have been configured.
`all`	All settings, whether configured or not.

Level The configurator schema requires each service and each of its settings to have an assigned level. Level enables users to distinguish between configuration data that is required for basic system functionality and configuration data that is used by some sites only for tuning specific configuration parameters.

`required`	Settings that must be set or the system will not function. This level is used primarily for services with settings that are not provided with a default value by the configuration template, usually because no reasonable default value exists. The config set will not validate if any required settings are skipped (i.e., left unset). Specify level `required` in a `cfgset` command to filter for required settings only.
`basic`	Settings that are likely to be used by most sites. If a `basic` setting is left unset, the template-provided default is used. Specify level `basic` in a `cfgset` command to filter for both basic and required settings.
`advanced`	Optional settings that are likely to be used only by advanced users to tune a service. If an `advanced` setting is left unset, the template-provided default is used. Specify level `advanced` in a `cfgset` command to filter for all settings: advanced, basic, and required.

Notes about filters:

- **Level filter**: although a given configuration service or setting is assigned only one level, the `level` option treats the levels as additive for the purpose of filtering. For example, the `cfgset create --level basic` command filters for services/settings that are level basic as well as those that are level required, while specifying level advanced includes services/settings of all three levels, and level required includes only level required services and settings.
- **Both filters**: when creating or updating a config set using configuration worksheets, the configurator ignores filters.

3.4 Create a Config Set

Choosing the best strategy for creating a config set depends on the circumstances ("when to use"):

Strategy	When to use	Rationale
Create a Config Set from Configuration Worksheets	when performing fresh installs, major upgrades, or any time there is a large amount of configuration data to set up	Worksheets can be generated, filled out offline with site-specific data by the appropriate staff, and then imported when needed.
Create a Config Set by Cloning	when there is already a config set with site-specific data and additional config sets are needed with minor variations (for partitions, alternate configurations, etc.), or when manually backing up a config set	Cloning is quick, and it is easy to interactively update the clone with needed variations.

Strategy	When to use	Rationale
Create a Config Set without Callbacks	when no hardware is attached to the XC system, as in some testing scenarios	Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content.
Create a Config Set Interactively	when configuring a smaller system with little configuration data to change	Setting all configuration values one at a time in response to a series of prompts or when selected from a menu can be very time-consuming.

These strategies all use the `cfgset` command. Use `cfgset create -h` for information about the create subcommand. See *Config Set Create/Update Process* on page 13 for an outline of the process followed by `cfgset` each time the `create` or `update` subcommand is used.

Note that when the `create` subcommand is used in any of these strategies (except cloning), it is necessary to specify the config set type for any type other than the default `cle`. Most of the following `create` procedures omit `--type` because they are for config sets of type `cle`.

REMEMBER: Run `cfgset` as root.

CAUTION: Boot failure possible if using `cfgset` under certain conditions.

The `cfgset create` and `cfgset update` commands always call pre- and post-configuration scripts. Some of these scripts require HSS daemons and other CLE services to be running. This can cause problems under these conditions:

- If `xtdiscover` is running, `cfgset` may hang or produce incorrect data that can result in system boot failure.
- If `xtbounce` is in progress or if the SMW is not connected to XC hardware, `cfgset` will fail.

In these circumstances, use the `--no-scripts` option with `cfgset create` or `cfgset update` to avoid running the scripts. Because using that option results in an invalid config set, remember to run `cfgset update` without the `--no-scripts` option afterwards, when circumstances permit, to ensure that all pre- and post-configuration scripts are run.

For more information on creating a config set using `--no-scripts`, see *Create a Config Set without Callbacks* on page 21

Create Backup Config Sets Automatically

If the `auto_clone` option in the IMPS configuration file (`/etc/opt/cray/imps/imps.json`) is enabled, the `cfgset create` and `cfgset update` commands will automatically clone a config set as a backup upon successful creation/update of the original config set. A failed operation will not create a backup.

The `autosave_limit` parameter in the IMPS configuration file determines how many clones will be retained. Config set backups are rotated with the oldest backup removed as a new backup is generated. Config set backups are saved with names of the form *`CONFIGSET`*`-autosave-`*`YYYY-MM-DDTHH:mm:SS`*, where *`CONFIGSET`* is the name of the original config set.

3.4.1 Create a Config Set from Configuration Worksheets

Prerequisites
This procedure has no prerequisites.

About this task
Use this procedure when performing fresh installs, major upgrades, or any time there is a large amount of configuration data to set up. To create a config set from configuration worksheets, use this process:

1. Generate the worksheets.
2. Copy the worksheets to a new location on the management node.
3. Edit the worksheets.
4. Import the worksheets.

The detailed steps of this procedure show an example of how to create config set `p0` of type `cle` (default) from configuration worksheets.

Note that the `cfgset` command is run as root.

Procedure

1. Generate new worksheets from configuration service packages installed on the system.

   ```
   smw# cfgset create --mode prepare p0
   ```

2. Locate the newly generated worksheets and copy them to a new location.

   ```
   smw# cfgset show --fields path p0
   p0:
     path: /var/opt/cray/imps/config/sets/p0

   smw# cp /var/opt/cray/imps/config/sets/p0/worksheets/* /some/edit/location
   ```

3. Edit the worksheets to customize them for this site.

   The system administrator typically distributes them to site staff members with knowledge about the services being configured so that they can edit the worksheets and enter appropriate values. Each worksheet is a YAML file that contains instructions on how to edit it; the basic idea is to locate the settings of interest, uncomment them, and either retain or change the default setting (if provided).

4. Import the completed worksheets using `cfgset update` or `cfgset create`.

   Import the completed worksheets by updating the config set created when the worksheets were generated originally or by creating an entirely new config set. The argument to the `--worksheet-path` option is a file glob to allow multiple worksheets to be imported in a single create/update operation. Full paths to single worksheets can also be used.

   - Import to the config set created with `--mode prepare` in step 1.

     ```
     smw# cfgset update --worksheet-path '/some/edit/location/*_worksheet.yaml' p0
     ```

   - Import to a new config set.

```
smw# cfgset create --worksheet-path '/some/edit/location/*_worksheet.yaml' \
 p0-new
```

REMEMBER: When importing worksheets using `cfgset` with the `--worksheet-path` option,

- Always add single quote marks around the worksheet path if a wildcard is used (e.g.,`*_worksheet.yaml`).
- Do not add mode, state, level, or service options; the configurator ignores them for worksheet import.
- The type of the config set must match the type of the worksheets being imported.

3.4.2 Create a Config Set by Cloning

Prerequisites

This procedure assumes that the config set to be cloned (the original) already exists.

About this task

Use this procedure when there is already a config set with site-specific data and additional config sets are needed with minor variations (for partitions, alternate configurations, etc.), or when manually backing up a config set. This procedure shows an example of creating config set `p0-new` by cloning it from existing config set `p0`. No callback scripts or configurator sessions occur when cloning a config set. The clone will have the same config set type as the original.

Note that the `cfgset` command is run as root.

Procedure

Create a clone using the `--clone` option.

```
smw# cfgset create --clone p0 p0-new
```

The configurator is not invoked when the `--clone` option is used, so no configurator session occurs, and no changes are made to the configuration data in the original config set.

3.4.3 Create a Config Set without Callbacks

Prerequisites

This procedure has no prerequisites.

About this task

Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content. Use this procedure when no hardware is attached to the XC system, as in some testing scenarios. This procedure shows an example of creating config set `global0` of type `global` from worksheets while skipping all callback scripts. The `--no-scripts` option can also be used when creating a config set interactively.

Note that the `cfgset` command is run as root.

Procedure

Create a config set without callbacks.

```
smw# cfgset create --no-scripts --worksheet-path \
'/some/edit/location/*_worksheet.yaml' --type global global0
```

CAUTION: Skipping callback script processing invalidates a config set. A config set cannot be considered validated unless it is updated successfully without the `--no-scripts` option. Update all config sets to run the callback scripts before using the config set with the system.

3.4.4 Create a Config Set Interactively

Prerequisites

This procedure has no prerequisites.

About this task

This procedure shows examples of creating config set `p0` of type `cle` interactively. For additional examples, use `cfgset create -h`.

Note that the `cfgset` command is run as root.

Procedure

Invoke the configurator in auto mode (default) or interactive mode.

- **Auto mode.**

  To be presented with all settings with state `unset` (default) and level `basic` (default) in all services in config set `p0`:

  ```
  smw# cfgset create p0
  ```

  To be presented with all settings (any state and any level) in all services in config set `p0`:

  ```
  smw# cfgset create --state all --level advanced p0
  ```

- **Interactive mode.**

  To display a menu of services in config set `p0` that have configuration settings with state `unset` (default) and level `basic` (default):

  ```
  smw# cfgset create --mode interactive p0
  ```

  To display a menu of all services (with settings of any state and any level):

  ```
  smw# cfgset create --mode interactive --state all --level advanced p0
  ```

3.5 Update a Config Set

Choosing the best strategy for updating a config set depends on the circumstances ("when to use"):

Strategy	When to use	Rationale
Update a Config Set Interactively	when one or more config sets require a few changes (e.g., cloned config sets that need to be adjusted for a particular purpose), when a software update introduces just a few new fields to configure, or to confirm that all required and basic settings have been set (very useful!)	Setting just a few configuration values one at a time in response to a series of prompts or when selected from a menu works well when there are just a few settings that need to be configured or updated.
Update a Config Set from Configuration Worksheets	when performing system upgrades and updates, or any time there is a large amount of configuration data to change	Worksheets can be generated, filled out offline with site-specific data by the appropriate staff, and then imported when needed.
Update a Config Set without Callbacks	when no hardware is attached to the XC system, as in some testing scenarios	Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content.
Rename a Config Set	when a config set needs to be renamed as well as updated, or just renamed	This could become necessary for a variety of reasons.
Update a Single Service in a Config Set	when setting up a new service, or when just one service requires modification	This can be done either interactively or with worksheets, so refer to those circumstances and rationales for the right strategy.

These strategies all use the `cfgset` command. Use `cfgset update -h` for information about the update subcommand. See *Config Set Create/Update Process* on page 13 for an outline of the process followed by `cfgset` each time the `create` or `update` subcommand is used.

CAUTION: Boot failure possible if using `cfgset` under certain conditions.

The `cfgset create` and `cfgset update` commands always call pre- and post-configuration scripts. Some of these scripts require HSS daemons and other CLE services to be running. This can cause problems under these conditions:

- If `xtdiscover` is running, `cfgset` may hang or produce incorrect data that can result in system boot failure.
- If `xtbounce` is in progress or if the SMW is not connected to XC hardware, `cfgset` will fail.

In these circumstances, use the `--no-scripts` option with `cfgset create` or `cfgset update` to avoid running the scripts. Because using that option results in an invalid config set, remember to run `cfgset update` without the `--no-scripts` option afterwards, when circumstances permit, to ensure that all pre- and post-configuration scripts are run.

For information on updating a config set using `--no-scripts`, see *Update a Config Set without Callbacks* on page 26

3.5.1 Update a Config Set Interactively

Prerequisites

This procedure assumes an existing config set needs to be updated.

About this task

Use this procedure when one or more config sets require a few changes (e.g., cloned config sets that need to be adjusted for a particular purpose), or to confirm that all required and basic settings have been set (very useful!). To update just one service in a config set, see *Update a Single Service in a Config Set* on page 26.

`cfgset` has two modes that initiate an interactive configurator session: `auto` (default) and `interactive`. This procedure shows examples of updating config set `p0` of type `cle` interactively in either mode. For additional examples, use `cfgset update -h`.

Note that the `cfgset` command is run as root.

Procedure

Invoke the configurator in auto mode (default) or interactive mode.

- **Interactive mode.**

  To display a menu of services in config set `p0` that have configuration settings with state `unset` (default) and level `basic` (default):

  ```
  smw# cfgset update --mode interactive p0
  ```

  To display a menu of services in config set `p0` that have configuration settings with level `required` and state `unset`:

  ```
  smw# cfgset update --mode interactive --level required p0
  ```

  To display a menu of all services in config set `p0`, use the broadest state and level filters:

  ```
  smw# cfgset update --mode interactive --state all --level advanced p0
  ```

- **Auto mode.**

  To confirm that all required and basic settings have been set (in which case, the configurator will not initiate an interactive session) or to be presented with all settings with state `unset` (default) and level `basic` (default) in all services in config set `p0`:

  ```
  smw# cfgset update p0
  ```

  For a discussion of common outcomes of this command, see *cfgset Troubleshooting Tips* on page 83.

  To be presented with all settings in config set `p0`, use the broadest state and level filters:

  ```
  smw# cfgset update --state all --level advanced p0
  ```

3.5.2 Update a Config Set from Configuration Worksheets

Prerequisites

This procedure assumes an existing config set needs to be updated.

About this task

Use this procedure when performing system upgrades and updates, or any time there is a large amount of configuration data to change. The configurator overwrites all data in a service with the contents of the worksheets specified on the command line. If a worksheet with stale data is used to update the config set, data loss may occur. To ensure that the worksheets used to update the config set are as up-to-date as possible, use this process:

1. Generate worksheets from the current config set.
2. Copy the worksheets to a new location on the management node.
3. Edit the worksheets.
4. Import the worksheets to the current config set.

The detailed steps of this procedure show an example of how to update config set `p0` of type `cle` (default) from configuration worksheets. To update just one service in a config set, see *Update a Single Service in a Config Set* on page 26.

Note that the `cfgset` command is run as root.

Procedure

1. Generate new worksheets from configuration service packages installed on the system and config set `p0`.

   ```
   smw# cfgset update --mode prepare p0
   ```

2. Locate the newly generated worksheets and copy them to a new location on the management node.

   ```
   smw# cfgset show --fields path p0
   p0:
     path: /var/opt/cray/imps/config/sets/p0

   smw# cp /var/opt/cray/imps/config/sets/p0/worksheets/* /some/edit/location
   ```

3. Edit one or more worksheets to make the needed changes.

   To edit the worksheets, open those with settings that need to be changed and make changes, as needed. Each worksheet is a YAML file that contains instructions on how to edit it.

4. Import the completed worksheets to `p0` using `cfgset update`.

   ```
   smw# cfgset update --worksheet-path '/some/edit/location/*_worksheet.yaml' p0
   ```

   The argument to the `--worksheet-path` option is a file glob to allow multiple worksheets to be imported in a single create/update operation. Full paths to single worksheets can also be used. The configurator will replace config set data with imported worksheet data only for services that have matching worksheets provided on the command line.

   REMEMBER: When importing worksheets using `cfgset` with the `--worksheet-path` option,

   - Always add single quote marks around the worksheet path if a wildcard is used (e.g.,`*_worksheet.yaml`).
   - Do not add mode, state, level, or service options; the configurator ignores them for worksheet import.
   - The type of the config set must match the type of the worksheets being imported.

3.5.3 Update a Config Set without Callbacks

Prerequisites
This procedure assumes an existing config set needs to be updated.

About this task
Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content. Use this procedure when no hardware is attached to the XC system, as in some testing scenarios. This procedure shows an example of updating config set `p0` of type `cle` interactively while skipping all callback scripts. The `--no-scripts` option can also be used when updating a config set from worksheets.

Note that the `cfgset` command is run as root.

Procedure

Update a config set without callbacks.

```
smw# cfgset update --no-scripts p0
```

CAUTION: Skipping callback script processing invalidates a config set. A config set cannot be considered validated unless it is updated successfully without the `--no-scripts` option. Update all config sets to run the callback scripts before using the config set with the system.

3.5.4 Rename a Config Set

Prerequisites
This procedure assumes an existing config set.

About this task
Use this procedure when a config set needs to be renamed or updated as well as renamed. The renaming operation follows the same basic configurator flow as a regular update but renames the config set prior to other processing. If auto-cloning is enabled, config set backups of the original config set will not be renamed. This procedure shows an example of renaming config set `p0`.

Note that the `cfgset` command is run as root.

Procedure

Rename a config set using the `update` subcommand with the `--rename` option.

```
smw# cfgset update p0 --rename p0.new
```

Note that the config set being operated on (`p0` in this example), does not have to be the last argument on the command line.

3.5.5 Update a Single Service in a Config Set

Prerequisites
This procedure assumes an existing config set.

About this task
Use this procedure when setting up a new service, or when just one service requires modification. This procedure provides examples of updating a single service at a time instead of the entire config set, and it can be done either interactively or using a configuration worksheet.

Procedure

Update a single service in config set `p0`.

- **Update interactively: use the `--service` option.**

  IMPORTANT: For a service with configuration template file `cray_example_config.yaml`, use only the `cray_example` portion on the command-line when specifying a single service.

  To display a menu of settings in the `cray_example` service in config set `p0` that are level `required` and any state (default for interactive mode when only one service is specified):

  ```
  smw# cfgset update --service cray_example --mode interactive \
  --level required p0
  ```

  To display a menu of all settings (with settings of any state and any level):

  ```
  smw# cfgset update --service cray_example --mode interactive \
  --level advanced p0
  ```

  To be presented with all settings (with settings of any state and any level):

  ```
  smw# cfgset update --service cray_example --state all --level advanced p0
  ```

- **Update with a worksheet: use the `--worksheet-path` option.**

  To update the service using a worksheet, use the `--worksheet-path` option instead of `--service`. Unlike the `--service` option, with the `--worksheet-path` option it is necessary to provide the full path to the worksheet for that service, which includes the `_worksheet.yaml` portion. The configurator will replace only the config set data that corresponds to the data in the worksheet being imported.

  ```
  smw# cfgset update --worksheet-path \
  /path/to/worksheets/cray_example_worksheet.yaml p0
  ```

3.6 Search a Config Set

Use Search to Locate Settings in a Config Set

The search subcommand is helpful when a user wants to view or change a configuration parameter (setting) but does not know which configuration template or worksheet contains it. To search for a configuration setting/field name or value, use the `cfgset search` command:

```
smw# cfgset search --term myvalue CONFIGSET
```

Search tips:

- To broaden a search, use multiple search terms (a logical OR).
- To narrow a search, use state and level filters.
- Unlike the `create` and `update` subcommands, the `search` subcommand has a default value of `all` for the state filter.

Here's an example that searches for the terms `c0-0c0s1n1` and `lus/` in settings of any level in config set p0:

```
smw# cfgset search --term c0-0c0s1n1 --term lus/ --level advanced p0
```

The configurator outputs highlighted dotted-path notation matches to the search term in a per-service report:

```
# 1 match for 'c0-0c0s1n1' from cray_scalable_services_config.yaml
#-------------------------------------------------------------------------------
cray_scalable_services_data.settings.scalable_service.data.tier1: c0-0c0s0n1, c0-0c0s1n1
# 1 match for 'lus/' from cray_node_health_config.yaml
#-------------------------------------------------------------------------------
cray_node_health_.settings.filesys_plugins.data.Default Filesystem.path: /lus/case1
...(more matches not included in example)
```

To output more information about the fields and values that match the search term(s), add the `--format full` command line option. This will display meta information about the setting in which the term was found, such as its level, state, and default value.

Note that the search subcommand does not search guidance text in the configuration templates and worksheets.

Use Search to Print out the Entire Config Set

To print out an entire config set, simply search the config set and omit the `--term` option. For example, to view all required fields that have not been set in config set p0, use the following command:

```
smw# cfgset search --level required --state unset p0
```

3.7 Retrieve or Modify Configuration Data Using the Command Line Interface

The `cfgset` command with subcommands `get` or `modify` can be used to retrieve and set configuration data values from the command line interface to the configurator tool, respectively. Data is retrieved and set on the command line using the same path to the data that is used in the configuration worksheets for the service being configured. The `modify` and `get` subcommands provide a quick, non-interactive means to modify and retrieve configuration field data. These subcommands are intended to be non-interactive, scriptable, and targeted. As a result, the workflow for changing configuration data using `cfgset modify` differs slightly from changing configuration data using the existing `cfgset create/update` commands.

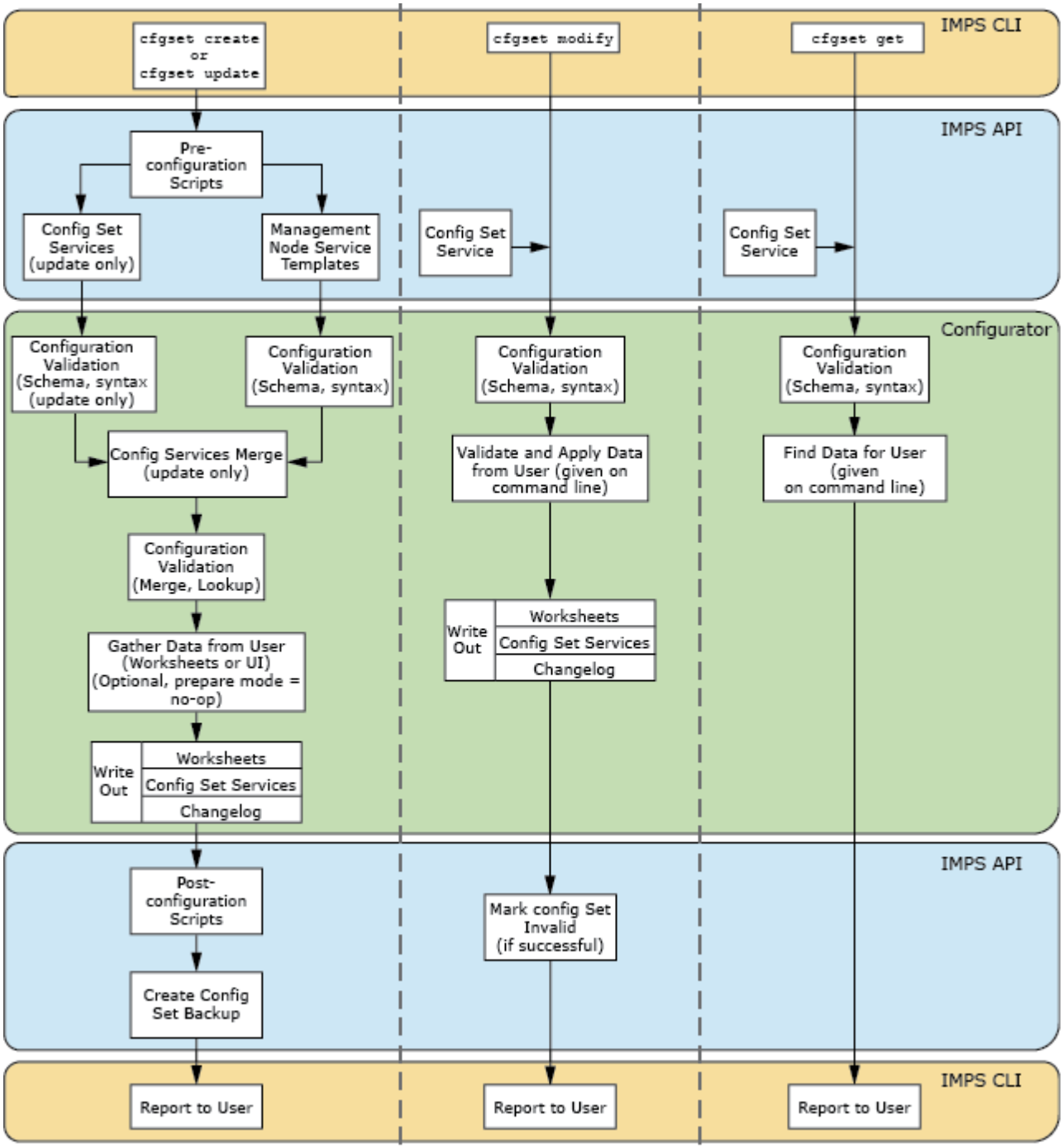
Figure 1 below compares the differences in workflow between using `cfgset create/update`, `cfgset modify`, and `cfgset get`. A few important workflow changes to note:

- `cfgset modify` does not invoke pre- and post-configuration scripts; therefore, a successful modification of configuration data using this command will result in the config set being marked invalid. Remember to run `cfgset update` afterwards, when circumstances permit, to ensure that all pre- and post-configuration

scripts are run. Note that `cfgset get` only reads data from the config set and thus will not mark a config set as invalid.

- The configurator will create a changelog file when invoked via `cfgset modify` of the same format as is done when modifications are made via `cfgset update/create`.
- The configurator will not search for management node templates on the SMW and thus will not attempt to merge templates with the config set service counter-parts. `cfgset modify` and `cfgset get` will only attempt modifications and retrievals of data from the config set services and their existing versions. `cfgset update --mode prepare` (or equivalent) must be used to merge template changes into the config set before modification. As a result, no merge validation will occur when running `cfgset modify` or `cfgset get`.

Figure 4. Comparing `cfgset` Workflows



Valid `path` Values

Valid paths for `path` on the command line for both `modify` and `get` interfaces are as follows (with examples from current CLE configuration services):

- Class-scoped settings fields:
  `cray_scalable_services.settings.scalable_service.data.tier1_groups`

- Multival-scope settings: `cray_node_groups.settings.groups.data`
- Embedded multival-scope settings: `cray_net.settings.hosts.data.bootnode.interfaces`
- Multival-scoped top-level settings entries: `cray_node_groups.settings.groups.data.all_nodes`
- Multival-scoped embedded settings entries: `cray_net.settings.hosts.data.bootnode.interfaces.network`
- Service enabled status: `cray_boot.enabled`
- Service inheritance status: `cray_multipath.inherit`

Configurator schema meta paths are not supported. This includes all entries under the `configurator` key in the configuration service files.

3.7.1 Use `cfgset get` to Retrieve Config Set Data

The `cfgset get` interface provides access to configuration data values relative to the path given to the command line. Paths denoting the full schema path to a data field will provide the value for the entered field. All output from `get` for successful retrievals will be printed to STDOUT and all error output will be printed to STDERR. Output which contains multiple values such as list data or lists of fields will print the values one per line. When the output includes empty or `null` values, the output will be as follows:

- Zero-length string: The output will be a new line `\n`.
- Zero-length list: The output will not return any lines.
- `null` value: The special identifier `__NULL__` will be output to delineate the value to help differentiate the value from more common string representations such as `null`.
- Boolean values will be returned as either `true` or `false`.

Generally, output from `cfgset get` for list fields redirected to a file will be usable as file input to `cfgset modify`. This does not apply to output that lists field names in a setting, as this may produce multiple values.

This example demonstrates querying the *`p0`* config set for the value of *`cray_ssh.enabled`*. The command interface returns the setting's value, in this case `true`:

```
smw# cfgset get cray_ssh.enabled p0
true
```

This example demonstrates querying the same config set for *`cray_alps.settings.common.data.xthostname`* and *`cray_net.settings.networks.data.hsn.ipv4_network`*:

```
smw# cfgset get cray_alps.settings.common.data.xthostname p0
cray
smw# cfgset get cray_net.settings.networks.data.hsn.ipv4_network p0
10.128.0.0
```

3.7.2 Use `cfgset modify` to Edit Config Set Data

The `cfgset modify` interface provides a method for manipulating data within a config set using the command line interface. Successful modifications to configuration data via a call to `modify` will not produce any output to

STDOUT; all error output wil be printed to STDERR. Error output from `modify` will use the existing error messages generated by the configurator for field validation errors. These errors are printed out during `cfgset update` and `cfgset validate` when validation is being performed for those actions. Errors related to improper or unknown PATH input to `cfgset modify` or `cfgset get` will print out the bad path as well as an indication of what the problem may be. For example, if a simple typo is on the element of the path (`boot_node` instead of `bootnode`):

```
smw# cfgset modify -s c0-0c0s0n1 cray_net.settings.hosts.data.boot_node.hostid p0
Error: could not modify 'p0': path=cray_net.settings.hosts.data.boot_node.hostid
Path entry 'boot_node' not found in schema.
```

The subcommand `modify` supports the following arguments:

Table 1. `cfgset modify` Optional Arguments

Argument	Description
`-s` *VALUE*, `--set` *VALUE*	Set/replace existing *VALUE* (non-`list` fields)
`-r` *VALUE*, `--remove` *VALUE*	Remove *VALUE* (`lists` fields only), or remove an entry named *VALUE* from a multival settings (multival settings only)
`--remove-file FILE`	Remove value(s) found in `FILE` (`lists` fields only) or remove entry/entries found in `FILE` from a multival setting (multival settings only)
`-a` *VALUE*, `--add` *VALUE*	Add/append *VALUE* (`lists` fields only), create new multival entry (multival settings only)
`--add-file FILE`	Add/append value(s) found in `FILE` (`lists` fields only) or create a new entry/entries found in `FILE` (multival settings only)
`-x, --clear`	Remove all entries (`lists` fields) or set value to empty (string-type fields) or set value to `null` (numeric/boolean fields)
`-d, --delete`	Delete all or single entries from a multival setting field.
`-h, --help`	Show all available arguments

The following example demonstrates retrieving the current value for `cray_net.enabled` using `cfgset get`, using `cfgset modify --set` to modify the existing value, and retrieving the value again to verify the change:

```
smw# cfgset get cray_net.enabled p0
false
smw# cfgset modify --set true cray_net.enabled p0
smw# cfgset get cray_net.enabled p0
true
```

This example demonstrates using `cfgset modify --add` to add a new node group called *test_nodes* to `cray_node_groups.settings.groups.data` in the `p0` config set, then using `cfgset get` to view the current node groups:

NOTE: A new multival setting entry is created by adding a multival key to the setting for both top-level and embedded multival settings. When the entry is created, all non-key fields will be populated with their default values.

```
smw# cfgset modify --add test_nodes cray_node_groups.settings.groups.data p0
smw# cfgset get cray_node_groups.settings.groups.data p0
compute_nodes
service_nodes
smw_nodes
boot_nodes
sdb_nodes
login_nodes
all_nodes
tier2_nodes
test_nodes
```

Multiple entries can be added or removed at a time using `--add` or `--remove`:

```
smw# cfgset modify --add c0-0c0s1n3 --add
c1-1c1s1n3 cray_node_groups.settings.groups.data.test_nodes.members p0
```

Use `--add-file` *FILE* or `--remove-file` *FILE* to add the values found in a text file (one value per line) to a setting:

```
smw# vim /tmp/test_nodes.txt
c2-2c2s1n3
c3-3c3s1n3
~
smw# cfgset modify --add-
file /tmp/test_nodes.txt cray_node_groups.settings.groups.data.test_nodes.members p0
smw# cfgset get cray_node_groups.settings.groups.data.test_nodes.members p0
c0-0c0s1n3
c1-1c1s1n3
c2-2c2s1n3
c3-3c3s1n3
```

The `--clear` option can be used to set a value to `NULL` in a boolean or numeric (integer or float) field, set a value to empty in a string-type field, or remove all entries in a list field:

```
smw# cfgset modify --clear
cray_node_groups.settings.groups.data.test_nodes.members p0
smw# cfgset get cray_node_groups.settings.groups.data.test_nodes.members p0
smw#
```

To delete an entry from a multival setting field, use `--delete`. This example demonstrates deleting the `test_nodes` entry followed by deleting all entries in `cray_node_groups.settings.groups.data`:

```
smw# cfgset modify --delete cray_node_groups.settings.groups.data.test_nodes p0
smw# cfgset get cray_node_groups.settings.groups.data p0
compute_nodes
service_nodes
smw_nodes
boot_nodes
sdb_nodes
login_nodes
all_nodes
tier2_nodes
smw# cfgset modify --delete cray_node_groups.settings.groups.data p0
```

```
smw# cfgset get cray_node_groups.settings.groups.data p0
smw#
```

3.8 Manually Edit Configuration Files

Cray recommends using only the configurator to modify data in config set configuration templates. Configuration worksheets and the available configurator user interfaces are the proper means to add, remove, and modify configuration data. However, configuration content is available in the config set directory for viewing or editing if required. Since the configurator not only manages configuration data, but also the configuration state of the data, users must take care that manual edits change both data and configuration status correctly to mimic the action of the configurator during `cfgset create`, `cfgset modify`, and `cfgset update` operations. For more information on how a setting is marked as configured, see *Use Filters to Choose What to See with the Configurator* on page 17. Manual edits to configuration data and status are not known to the configurator and therefore are not reflected in the changelog files produced when the configurator is invoked to modify configuration data.

If content has been modified by a user or process outside of the `cfgset` command and its subcommands, care should be taken to ensure that the content within the config set is viable for downstream consumers of this data, as is done with Cray-provided tools. Specifically, if configuration templates are modified in the config set, the config set should be updated to run the pre- and post-config script hooks with the `cfgset update --mode prepare` command. The config set should also be revalidated with the `cfgset validate` command after modification. This will ensure basic syntax, schema, and error checking of the configuration data.

3.8.1 Manually Edit Service Enable/Inherit Data

Procedure

1. Open the configuration data YAML file using a text editor and make the desired changes to the `enabled` and `inherit` fields located at the top of the YAML file. This example uses the `cray_time` configuration service in `cray_time_config.yaml`:

   ```
   cray_time:
       enabled: true
       inherit: false
       settings:
           ...
   ```

2. Locate the `configurator` field that contains the configuration metadata for the top-level service data fields that were modified in the previous step. This section is located after all service settings, near the end of the YAML file. Change the `configured` field to `true` if the value of the `enabled` field was modified. Similarly, change the value of the `configured_inherit` field to `true` if the value of the `inherit` field was modified. If the value of either field was already set to `true`, no action is required:

   ```
   ...
       configurator:
           ...
           configured: false          # <- set configuration status for 'enabled'
   field
           configured_inherit: false  # <- set configuration status for 'inherit'
   ```

```
field
        ...
```

3.8.2 Manually Edit Class-Scoped Setting Data

Procedure

1. The configurator represents simple key-value pairs of data in settings called class-scoped settings. To properly manually edit class-scoped data, first verify that the setting is of `scope_type: class` by checking the `scope_type` field in the setting to be modified:

```
cray_time:
    ...
    settings:
        service:
            ...
            configurator:
                ...
                scope_type: class
                ...
```

2. If the value of the `scope_type` field is `multival`, see *Manually Edit Multival-Scoped Setting Data* on page 35. To manually edit a `scope_type: class` data field, first edit the data field itself. This example demonstrates editing the `timezone` field in `cray_time`:

```
cray_time:
    ...
    settings:
        service:
            data:
                ...
                timezone: US/Central    # <- modify as required
            ...
```

3. Edit the configuration status of the modified field. Locate the configurator metadata section for the modified field and set the value of the `configured` field to `true`. In this example, since the `timezone` field was modified, the corresponding configuration metadata section will be set to `true`:

```
cray_time:
    ...
    settings:
        service:
            ...
            configurator:
                argspec:
                    ...
                    timezone:
                        allow_none: false
                        configured: false     # <- set to true
```

3.8.3 Manually Edit Multival-Scoped Setting Data

Procedure

1. The configurator represents multiple entries of key-value pairs of data in settings called multival-scoped settings. To properly manually edit a multival-scoped data, first verify that the setting is of `scope_type: multival` by checking the value of the `scope_type` field in the setting to be modified:

```
cray_simple_shares:
    ...
    settings:
        NFS:
            ...
            configurator:
                ...
                scope_type: multival
                ...
```

2. If the value of the `scope_type` field is `class`, see *Manually Edit Class-Scoped Setting Data* on page 35. To manually edit a `scope_type: multival` data field, first edit the data fields for the desired entries in the `NFS` setting `data` key. This example demonstrates editing `fs_mount_opt` in the `NFS` setting in `cray_simple_shares`. For instance, if changes are desired in the `/var/opt/cray/imps` and `/non_volatile` entries, find the entries in the data and modify the value of the `fs_mount_opt` field:

```
cray_simple_shares:
    ...
    settings:
        ....
        NFS:
            data:
            ...
            -   key: /var/opt/cray/imps
                ...
                fs_mount_opt: ro    # <- modify as required
                ...
            -   key: /non_volatile
                ...
                fs_mount_opt: ''    # <- modify as required
                ...
            ...
```

3. Edit the configuration status of the modified field. Locate the configurator metadata section for the modified field (in this example, `fs_mount_opt`) and locate the `configured` and `unconfigured_keys` fields:

```
cray_simple_shares:
    ...
    settings:
        ....
        NFS:
            ...
            configurator:
                argspec:
                    ...
                    fs_mount_opt:
                        ...
                        configured: true
                        unconfigured_keys:
```

```
                        - /var/opt/cray/imps
                        - /non_volatile
                        ...
```

With multival-scope settings, the configurator keeps track of the configuration status of each individual entry in the setting. In this case, the `configured` field contains the overall configuration status of the `fs_mount_opt` portion of all entries. Exceptions to the status are maintained in the `unconfigured_keys` field. The table below describes the scenarios for how configuration status is determined by the configurator for individual fields in multival-scoped settings

`configured` Field	**`unconfigured_keys` Field**	**Configuration Status**
`false`	N/A	All entries of the field are considered not configured, regardless of the entries in the `unconfigured_keys` field.
`true`	Empty List	All entries of the field are considered to be configured.
`true`	Non-Empty List	All entries except the entries with the keys in the `unconfigured_keys` list are considered to be configured.

For this example, the `/var/opt/cray/imps` and `/non_volatile` entries are removed from the `unconfigured_keys` tag, since these entries were modified in step 2. The `configured` tag is then set to `true` as seen in the screenshot below.

CAUTION: The value of the `unconfigured_keys` field must always be of `list` type, as understood by YAML. If all elements are removed from the value of the `unconfigured_keys` field, the value of the field should be set to an empty list by placing brackets in the value of the field: `unconfigured_keys: []`

```
cray_simple_shares:
    ...
    settings:
        ....
        NFS:
            ...
            configurator:
                argspec:
                    ...
                    fs_mount_opt:
                        ...
                        configured: true
                        unconfigured_keys: []
                        ...
```

For multival settings nested within other multival settings, the `unconfigured_keys` field will contain period-delimited keys. For example, the `unconfigured_keys` list entry for `cray_net.settings.hosts.data.bootnode.interfaces.hsn_boot_alias.ipv4_address` would be as follows in the configurator file `cray_net_config.yaml`:

```
cray_net:
    ...
    settings:
```

```
...
    hosts:
        ...
        configurator:
            argspec:
                ...
                interfaces:
                    argspec:
                        ...
                        ipv4_address:
                            ....
                            configured: true
                            unconfigured_keys:
                            - bootnode.hsn_boot_alias
                            ...
```

3.9 Validate a Config Set and List Validation Rules

It is important to validate any config set that has been modified, because there is currently no mechanism to prevent the system from trying to use an invalid config set. Validation is useful for determining if the config set is minimally viable for use with the system it is intended to configure.

> **IMPORTANT:** Validation ensures that a config set passes all rules stored on the system. A validated config set does not necessarily equate to a config set with configuration data that will result in a properly configured system.

When validating a config set, the configurator checks the following:

- Config set has the proper directory structure and permissions.
- All configuration templates have correct YAML syntax.
- All configuration templates adhere to the configurator schema.
- All fields of type `lookup` reference values and settings that exist in the available configuration services.
- All level `required` fields in enabled services are configured (i.e., their state is `set`).
- Pre-configuration and post-configuration callback scripts ran successfully during the latest config set update.
- `cfgset validate` has run all validation rules installed on the system.

Validate a Config Set with the `validate` Command

To validate a config set, use the `cfgset validate` command:

```
smw# cfgset validate p0
```

The `cfgset validate` command runs all rules installed on the system. Users may specify which rules to include or exclude by using the rules file in `/etc/opt/cray/imps/rules.yaml`.

The `--no-rules` subcommand can be used to prevent the `cfgset` from executing any validation rules against the config set. All other validation checks will be done.

```
smw# cfgset validate --no-rules p0
```

> **NOTE:** Using the `--no-rules` option will not invalidate a config set, unlike `cfgset create/update --no-scripts` command behavior.

The `--include-rule` subcommand specifies a rule name to execute to validate the config set. Multiple `--include-rule` declarations can be made. Rules included via this parameter supersede rules specified in the rules file (`/etc/opt/cray/imps/rules.yaml`). Included rules supersede all excluded rules as well.

```
smw# cfgset validate --include-rule INCLUDE_RULE p0
```

The `--exclude-rule` subcommand specifies a rule name to skip when validating the config set. Multiple `--exclude-rule` declarations can be made. Rules excluded via this parameter supersede rules specified in the rules file (`/etc/opt/cray/imps/rules.yaml`).

To validate the resulting configuration services after a merge of the service packages with the config set content, add the `--merge` option.

```
smw# cfgset validate --merge SERVICE_PACKAGE
```

List Validation Rules with the `list-rules` Command

Use the `cfgset list-rules` command to list the validation rules for a given config set:

```
smw# cfgset list-rules p0
Listing the rules for the config set.

Rules:

- name: sdb.cray_sdb.CraySDBEnabled
  description: The cray_sdb service must be enabled.
  location: /opt/cray/imps_config/sdb/default/configurator/rules/cray_sdb.py

- name: sdb.cray_sdb.SDBGroupsNodeCheck
  description: The cray_sdb service must only configure tier1 and/or tier2 nodes as
SDB nodes.
  location: /opt/cray/imps_config/sdb/default/configurator/rules/cray_sdb.py
```

The `--service` *SERVICE* subcommand can be used to list the rules that apply to a specified service. The `--service` subcommand should not be used with the `--name` subcommand.

```
smw# cfgset list-rules --service cray_boot p0
Listing rules for the cray_boot service.

Rules:

- name: system-config.cray_boot.BootGroupsNodeCheck
  description: The cray_boot service must only configure tier1 and/or tier2 nodes
as boot nodes.
  location: /opt/cray/imps_config/system-config/default/configurator/rules/
cray_boot.py

- name: system-config.cray_boot.BootNodeGroupsNotEmpty
  description: The cray_boot service must set at least one node as the boot node.
  location: /opt/cray/imps_config/system-config/default/configurator/rules/
cray_boot.py
```

The `--name` *NAME* subcommand can be used to limit the output of the rule listing to a specified service for the given config set. The `--name` subcommand should not be used with the `--service` subcommand.

```
smw# cfgset list-rules --name system-config.cray_storage.CrayStorageEnabled p0

- name: system-config.cray_storage.CrayStorageEnabled
```

```
  description: The cray_storage service must be enabled.
  location: /opt/cray/imps_config/system-config/default/configurator/rules/
cray_storage.py
```

3.10 Remove a Config Set

To remove a config set, use the `cfgset remove` command:

```
smw# cfgset remove CONFIGSET
```

for a config set named *CONFIGSET*. The `remove` subcommand also accepts wildcards for removing multiple config sets at a time:

```
smw# cfgset remove p0.test*
```

This is especially helpful if config set auto-cloning/backup is enabled and removing all of the backups of a config set is desired. See the `cfgset` man page for a description of its subcommands and options and some examples of each, or use `cfgset remove -h` for help.

3.11 Back Up or Restore User, Group, and Permissions Information Files

Administrators can back up and restore the user, group, and permissions information of files contained under the cfgset directories using the `cfgset_export_perms.py` and `cfgset_restore_perms.py` scripts, respectively. These scripts are normally found in `/opt/cray/imps/default/etc/` and preserve the user ID (UID), group ID (GID), and file permissions of the specified directory. The default directory to be backed up or restored to is the current working directory. This is typically a cfgset directory in `/var/opt/cray/imps/config/sets/`. The output of `cfgset_export_perms.py` can be checked in to a source control program, such as Git, enabling a site to safely back up and restore config sets to a central source control server.

Back Up Using `cfgset_export_perms.py`

The `cfgset_export_perms.py` script will create a backup file `cfgset_permissions_metadata` that is placed in the current working directory. Any time a change is made to a config set, `cfgset_export_perms.py` should be run to capture the addition or removal of any files and permission or ownership changes. The following example demonstrates backing up the user, group, and permissions information located in the `/var/opt/cray/imps/config/sets/p0` directory and viewing the backup file `cfgset_permissions_metadata_p0` created. Note the use of the `--output` subcommand to specify the backup file name:

```
smw# ./cfgset_export_perms.py --output
cfgset_permissions_metadata_p0 /var/opt/cray/imps/config/sets/p0
smw# ls
cfgset_export_perms.py  cfgset_restore_perms.py  cfgset_permissions_metadata
smw# vim cfgset_permissions_metadata_p0
"",0,0,"0755"
".imps_ConfigSet_metadata",0,0,"0644"
```

```
"ansible",0,0,"0755"
"dist",0,0,"0755"
"dist/compute-preload.cray",0,0,"0660"
"dist/login-preload.cray",0,0,"0660"
"dist/README",0,0,"0660"
"dist/simple_examples",0,0,"0755"
"dist/simple_examples/sample_config_tasks.yaml",0,0,"0660"
"dist/simple_examples/roles",0,0,"0755"
"dist/simple_examples/roles/example",0,0,"0755"
"dist/simple_examples/roles/example/defaults",0,0,"0755"
"dist/simple_examples/roles/example/defaults/README",0,0,"0660"
"dist/simple_examples/roles/example/files",0,0,"0755"
"dist/simple_examples/roles/example/files/README",0,0,"0660"
"dist/simple_examples/roles/example/handlers",0,0,"0755"
"dist/simple_examples/roles/example/handlers/README",0,0,"0660"
"dist/simple_examples/roles/example/meta",0,0,"0755"
...
```

Restore Using `cfgset_restore_perms.py`

The `cfgset_restore_perms.py` script will restore the user, group, and permissions information contained in a backup file given to the script. By default, the script will search for a backup file named `cfgset_permissions_metadata` in the current working directory and restore permissions to the working directory. A different backup file to read from or a different directory to restore permissions to may optionally be specified. The following example demonstrates restoring permissions to `/var/opt/cray/imps/config/sets/p0` using the `cfgset_permissions_metadata_p0` backup file. Note the use of the `--input` subcommand to specify the backup file to read from:

```
smw# ./cfgset_restore_perms.py --input
cfgset_permissions_metadata_p0 /var/opt/cray/imps/config/sets/global
```

Integrate Scripts as Git Hooks

The `cfgset_export_perms.py` can be specified as a pre-commit hook. The `cfgset_restore_perms.py` script can be specified as a post-merge or post-checkout hook. It is recommended that a separate Git repo be configured for each config set. To establish a Git repo in the cfgset directory, follow the steps below:

1. If Git is not currently installed, install Git using Zypper:

   ```
   # zypper in git
   ```

2. Establish the Git directory in each config set directory that is to be backed up:

   ```
   smw# cd /var/opt/cray/imps/config/sets/p0
   smw# git init
   smw# git add -A
   smw# git commit -m "Initial check-in of P0 config set"
   ```

3. Optionally, establish any site-specific Git remotes.

To designate either script as a Git hook, follow these steps:

1. Copy the desired script(s) to the `.git/hooks` directory of the Git repository.
2. To use `cfgset_export_perms.py` as a pre-commit hook, rename the copied script to: `pre-commit`. To use `cfgset_restore_perms.py` as a post-checkout or post-merge hook, rename the copied script(s) to either: `post-checkout` or `post-merge`. It is suggested that `cfgset_restore_perms.py` be installed as both a post-checkout and post-merge hook, as the post-checkout commit hook is only called when doing a

`git checkout` operation. Likewise, the post-merge commit hook is called only when performing a `git pull` or similar operation. Note that to use `cfgset_restore_perms.py` as both a post-checkout and post-merge script, the script must be copied to the `.git/hooks` directory twice, with one copied script renamed as `post-checkout` and the other copied script renamed as `post-merge`.

3. Ensure all scripts are executable.

4 Configurator User Interface

The configurator provides a text-based user interface for setting configuration data in an interactive session. It also provides help on how to use the UI: within the configurator interface, enter **?** at any point to display a contextual help menu with input, navigation, and control options appropriate for the current screen.

For a quick introduction to using the configurator user interface, see *Tips for Configurator Interactive Sessions* on page 43.

For a more in-depth look at data types, menus, and basic operations, see the following topics, which are based on `cray_example`, an example service created for illustrative purposes only.

- *Configurator Data Types and How to Set Them* on page 47
- *Configurator Screens and Menus* on page 58 (includes a description of navigation controls)
- *Basic Configurator UI Operations* on page 62 (how to change filters, switch modes, view changes, and exit a session)

4.1 Tips for Configurator Interactive Sessions

When a user invokes `cfgset` in `auto` or `interactive` mode to create or update a config set, `cfgset` invokes the configurator to initiate an interactive session with the user. The configurator provides command help to aid users in navigating the tool and adding/updating configuration data. These tips supplement that help.

Know the difference between the two "interactive" modes

Interactive mode and auto mode can both result in a configurator interactive session, but their uses and behaviors are quite different.

auto mode

Helpful for verifying that all desired settings have been set.

Auto mode initiates an interactive session when there are one or more settings in the config set that meet state and level filtering criteria. Those settings are presented one at a time, and when all have been presented, the configurator exits the session.

interactive mode

Helpful for seeing the "big picture" and having more control over which services/settings are presented for configuration.

Interactive mode always initiates an interactive session. It provides two tiers of menus from which users can select one or more services/settings to drill down and configure just what is needed. The configurator presents the selected settings one at a time, as in auto mode, but when all selected settings have been presented, it returns the user to the menu from which the selection was made.

- Service Configuration List Menu (or Service List Menu) lists the services in the config set

- Service Configuration Menu (or service menu) lists the settings in a particular service

Filter wisely

Level and state filters determine what the configurator displays to users: what is included in the menu of services/settings for selection in interactive mode, and what setting fields are presented automatically for configuration in auto mode. The filters can be specified on the command line when invoking `cfgset`, and they can be changed in interactive mode. If not specified, they default to level `basic` and state `unset` (exception: for interactive mode, if a single service is specified, the default state is `all`).

In interactive mode, the configurator populates the Service List Menu with only those services that meet state and level filtering criteria; both filters can be switched to different values on this menu screen. In the case of a service menu, the configurator populates it with only those setting fields that meet level filtering criteria (shows all states); level can be switched on this menu screen, but state cannot. Just for fun, cycle through all levels/states, noting how level affects which services appear in the list, while state affects the status displayed for each service.

> **TIP:** If the desired service/setting is not visible in an interactive-mode menu, simply switch level.

In auto mode, the configurator presents only those setting fields that meet state and level filtering criteria. There is no opportunity to switch filter values in auto mode, except by first switching to interactive mode.

> **TIP:** A good way to confirm that all basic settings have been set is to run `cfgset update p0` (where `p0` is the config set name), which defaults to auto mode, level `basic`, and state `unset`. If the configurator does not present any settings, it means that no `basic` or `required` settings are unset.

How to switch states and levels (interactive mode only):

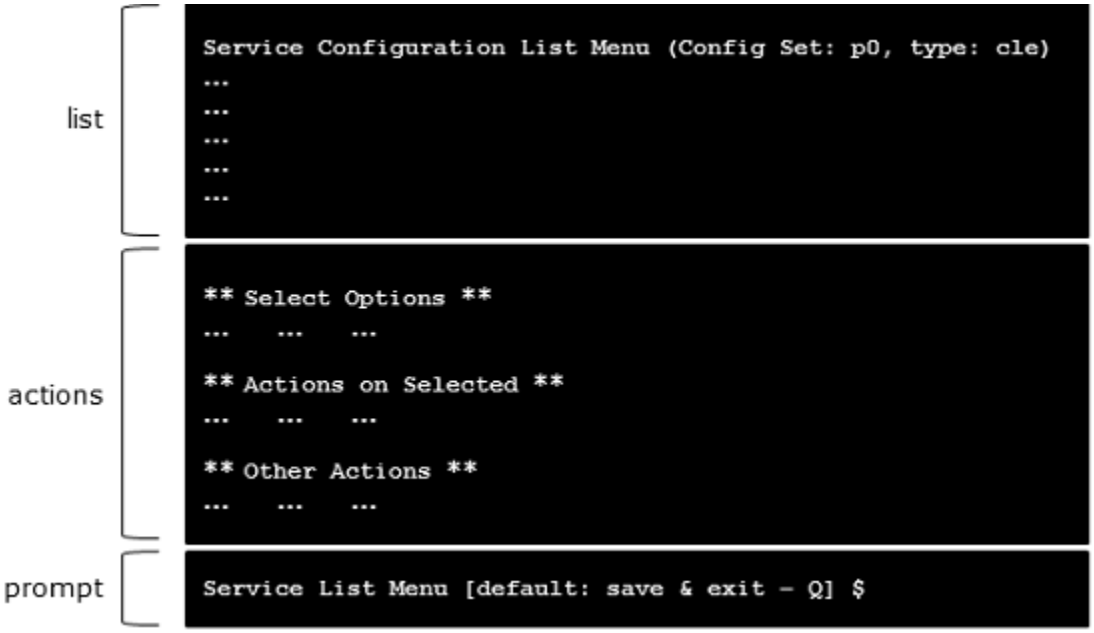
switch states	Enter **s** at the configurator prompt to switch from the current state to the next one: `unset`→`set`→`all`. To see all services/settings with the specified level, enter **s** until `state=all` displays in the menu header.
switch levels	Enter **l** (lowercase L) at the configurator prompt to switch from the current level to the next one: `basic`→`advanced`→`required`. To view all services/settings with the specified state, enter **l** until `level=advanced` displays in the menu header.

To see all possible services/settings, switch to `state=all` and `level=advanced`.

Get familiar with menus in interactive mode

The Service List Menu and all service menus have the same three-section layout: a list of services/settings, actions the user can take, and a prompt.

Figure 5. Sections of Interactive-Mode Menus



list The menu name, config set name, and config set type are shown at the top of the list section. This section is helpful for seeing which services still have unconfigured settings (status column—see what changes when state is switched) and for selecting which service(s) to configure or reconfigure.

In a service menu, the list items are configuration settings for that particular service, filtered by level only (state is set to `all` and cannot be switched). This list is helpful for seeing the current state and value of the settings and for selecting which setting(s) to set or change.

actions These three submenus show all commands currently available. Always use an action from the **Select Options** submenu before using any from the **Actions on Selected** submenu. Items in the **Other Actions** submenu can be used at any time (with the obvious exceptions of the exit commands **Q** and **x**, because when one of those is used, the configurator exits the interactive session).

Select Options	Actions that select one or more services/settings from the list. The selected services/settings are the only ones that can be acted upon. Once selected, an asterisk appears in the **Selected** column next to the item and its font color changes.
Actions on Selected	Actions that can be used on the selected service(s) or setting(s); a selection must be made first. Shows in parentheses how many items have been selected. A few of these actions, like toggle whether a service is enabled and toggle whether it inherits setting values from the global version of its template (applies to only a few services) move to the **Other Actions** submenu on service menu screens.
Other Actions	Actions that can be used on all services/settings or on the current configurator session. The most commonly used are the filter switches and help (?).

prompt The prompt shows which menu is active and what the default action is. Before a selection is made, the default action is to save and exit (as shown in previous figure). When a selection is made, the default action is to configure the selected service(s) or setting(s), and the prompt changes to

```
MENU_NAME [default: configure – C] $
```

Note that accepting this default action (or entering **C**) displays the configuration setting screen for the first selected setting.

Get familiar with configuration setting screens

A configuration setting screen shows users information about the setting field to be configured (default/current values, data type, level, current state, etc.) and enables the user to navigate among setting fields, enter/change field values, and switch to interactive mode. The configuration setting screen is displayed when a user makes a selection and enters **C** in interactive mode, or when a setting matches state and level filters in auto mode. Configuration setting screens have a prompt that is packed with useful information. Consider this example of a prompt:

```
cray_lmt.settings.lmt_database.data.database_fstype
[<cr>=set 'ext3', <new value>, ?=help, @=less] $
```

The first line is the full name of the setting field being presented (this is the same as the corresponding entry in the configuration worksheet for this service). The part that precedes `.settings.` is the service name (`cray_lmt`, the Lustre Monitoring Tool service, in the example), and the part that follows is the setting field being presented. In the example, the setting is `lmt_database` and the field to be set (one of several for that setting) is `database_fstype`.

The second line lists available commands. In the example, the default command (selected by pressing **Enter** or <cr>) sets the value to `ext3`, which is the default value provided in the configuration template for that service. If this setting field had already been configured with the value `ext3`, the default command would be `<cr>=keep 'ext3',` (`set` becomes `keep`). This list of available commands is not exhaustive: to see all possible options, enter **?** after the prompt, which will insert a context-sensitive menu of commands between the information section and the prompt.

Switch to interactive mode, as needed

When in a configuration setting screen, whether the user has arrived there by invoking `cfgset` in auto mode or by making a selection and entering **C** in interactive mode, it is possible to switch to interactive mode and display either the service menu (lists settings for a single service) or the Service List Menu (lists services in the config set).

switch from setting screen to a service menu

To switch to interactive mode and display the service menu, enter **^** at the configurator prompt. Example:

```
cray_node_health.enabled
[<cr>=keep 'true', <new value>, ?=help, @=less] $ ^
```

switch from setting screen to Service List Menu

To switch to interactive mode and display the Service List Menu, enter **^^** at the configurator prompt. This action can be taken only if `cfgset` was invoked for all services (as this is the default, this is true unless the `--service` or `-s` option was used). Example:

```
cray_node_health.enabled
[<cr>=keep 'true', <new value>, ?=help, @=less] $ ^^
```

Switch between menus in interactive mode, as needed

switch from Service List

When a service has been selected from the Service List Menu in interactive mode, enter **v** (view settings) to switch to the selected service's menu instead of taking the default action of

Menu to service menu	Configure (**C**). The **v** action is available if only a single service is selected. If multiple services are selected, **C** is the only action available. Example: `Service List Menu [default: configure - C] $ v`
switch from service menu to Service List Menu	To switch from a service menu to the Service List Menu, enter **^^** at the configurator prompt. This action can be taken only if `cfgset` was invoked for all services (as this is the default, this is true unless the `--service` or `-s` option was used). Example: `Node Health Service Menu [default: save & exit - Q] $ ^^`

When in doubt, jump out

It is better to leave a setting field unconfigured than set it to an incorrect value or 'none.' If unsure what the value should be or whether that setting field is needed, jump out using one of these methods:

- *Switch to interactive mode, as needed*.
- Skip to the next setting field: enter **>** at the configurator prompt.

Get help early and often

Enter **?** at the configurator prompt at any time to see a list of available commands. In interactive mode, this simply displays a verbose list of the same commands listed in the menu's three action submenus. However, in a configuration setting screen, entering **?** displays a context-sensitive menu of available commands not displayed elsewhere. Here is an example of the commands available in the context of configuring a multival setting in a service (multival settings are configured by adding/changing entries). Use the **?** command in configuration setting screens early and often to learn the available commands.

```
|--- Command Help
|   *     ++     - double view limit (currently 2)
|   *     --     - decrease view limit by half (currently 2)
|   *     *      - view all entries (no limit)
|   *     +      - add entries
|   *     <#>*   - change the <#> entry. Example: '2b*' selects sub-item b in entry 2
to change
|   *     <#>-   - delete the <#> entry. Example: '4-' deletes entry 4
|   *     d      - delete all entries in the list
|   *     <cr>   - accept the current value(s)
|   *     #      - set the value to its default
|   *     <      - go back to the previous setting
|   *     >      - skip and go to the next setting
|   *     ^      - Go to the 'cray_dvs' service menu (interactive mode)
|   *     ^^     - Go to the service list menu (interactive mode)
|   *     Q      - write out changes and exit the configurator
|   *     x      - revert all changes and exit the configurator
|   *     r      - refresh the screen
|   *     @      - toggle more/less info
|   *     ?      - show this help
```

4.2 Configurator Data Types and How to Set Them

Basic: String, Integer, Float, Boolean

To enter values for string, integer, and float data, enter the value of the field at the prompt, then press the **Enter** (<cr>) key. To accept the default value instead (if available), just press **Enter**. For booleans, valid values are `true` and `false`.

```
****** cray_example.settings.basic_example.data.basic_string_value *******
      basic_string_value -- Basic String Value
        This field is part of the basic_example setting in the cray_example
        service. It is a string value that only allows lowercase letters. The
        configurator will validate the value for this field that it is a string
        and matches the letters. Its default value is 'abc'.

        Default:       Current:
            abc            not configured yet

     Value: string, blank values not allowed, regex=^[a-z]*$
            level=basic, state=unset

     Inputs: <string> -- OR --  menu commands (? for help)

cray_example.settings.basic_example.data.basic_string_value
[<cr>=set 'abc', <new value>, ?=help, @=less] $
```

Advanced: Protected

For configuration data of type `protected`, which is often used for password fields, the configurator prompts the user to enter the value twice, and the entered value is not echoed to the terminal. The prompt changes slightly to indicate this different input mechanism.

```
cray_example.settings.basic_example.data.basic_protected_value
[+=modify, ?=help, @=less] $
```

To set or change a protected value (basic_protected_value in this example), press the **+** key to enter input mode, then enter the value of the field as directed.

```
cray_example.settings.basic_example.data.basic_protected_value
[+=modify, ?=help, @=less] $ +

Modify basic_protected_value (Ctrl-d to cancel, <cr> to set) $ << value entered >>
Re-enter value for basic_protected_value (Ctrl-d to cancel, <cr> to set) $ << value
entered >>
[...next screen rendered...]
```

Advanced: List

Some configuration data has type `list`, which accepts multiple entries for the value. For list data, the configurator prompt changes slightly to indicate this different input mechanism.

```
cray_example.settings.basic_example.data.basic_list_value
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $
```

To set or change a list value (`basic_list_value` in this example), enter **+** to enter input mode, then enter the value of the field as directed:

```
cray_example.settings.basic_example.data.basic_list_value
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $ +
Add basic_list_value (Ctrl-d to exit) $ first_value
Add basic_list_value (Ctrl-d to exit) $ second_value
Add basic_list_value (Ctrl-d to exit) $ << Ctrl-d >>
```

After **Ctrl-d** is pressed to exit input mode, the configurator redraws the field description with the proposed changes.

```
******* cray_example.settings.basic_example.data.basic_list_value *******

      basic_list_value -- Basic List Value
        This field is part of the basic_example setting in the cray_example
        service. It is a list value. The input mechanism is slightly different
        for protected values.

        Default:           Current:
            (none)             1) first_value
                               2) second_value

    Value: list, blank values not allowed
           level=basic, state=unset

    Inputs: menu commands (? for help)

|--- Information
|  *    2 entries added. Press <cr> to set.
|---

cray_example.settings.basic_example.data.basic_list_value
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $
```

The configurator adds an informational message about the input that was just added and lists the new values under `Current`. To actually set the list entries as the value of this field, press the **Enter** (<cr>) key.

> **IMPORTANT:** When finished editing list entries, always press **Enter** to set the entries as the value of the list field. If the save and exit command (**Q**) is issued before the entries are set, the configurator will not save the entries and will not write out the new data.

Advanced: Multival

Configuration data of type `multival` is used for entities that typically occur as multiple instances and each instance has a set of properties that need to be configured, such as network interfaces or DVS client mounts. A multival setting is basically a setting with one or more subsettings (entries), where each entry consists of a key (the entry name) and one or more fields of any data type (the fields do not need to have the same data type). One or more of the fields in a entry could be another multival!

As with list data, when a multival field is presented by the configurator, enter **+** to add a new entry. For each new entry, the configurator will present the multival key field to create the entry name, then cycle through the rest of the fields for the multival entry. The multival setting will be presented at the end for an opportunity to review, modify, add, or set new multival entries, as shown in this example. This multival setting has two entries defined, each of which has a key field (always a string), a string field, a protected field, and a list field.

```
********** cray_example.settings.multival_example **********

      multival_example
        Multiple hostnames can be configured in this service. For each
        hostname, a few simple values are available for further
```

```
        configuration.

    Configured Values:
        1) 'foo'
            a) basic_string_value: foovalue
            b) basic_protected_value: *** <hidden> ***
            c) basic_list_value:
                   foovalue1
                   foovalue2

        2) 'bar'
            a) basic_string_value: barvalue
            b) basic_protected_value: *** <hidden> ***
            c) basic_list_value:
                   barvalue1
                   barvalue2

    Inputs: menu commands (? for help)

|--- Information
|  *    Multiple 'multival_example' entries can be added using this menu
|---

cray_example.settings.multival_example
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $
```

IMPORTANT: As with list data, when finished editing multival entries, always press **Enter** to set the entries as the value of the multival setting. If the save and exit command (**Q**) is issued before the entries are set, the configurator will not save the entries and will not write out the new data.

Advanced: Lookup

The Cray Node Groups feature allows quickly switching between a field that needs to set its values as names of node groups and the node groups configuration service. The Cray Node Groups feature uses `lookup` fields to assign node group names to various other fields throughout the configuration set. Configuration `lookup` fields allow the user to specify values that are set in other fields and configuration services. `Lookup` fields act in a similar manner as `list` fields with the exception that only values that are set in the reference field are valid in the lookup field. The configurator specifies the extra options `!` and `v` when presenting a `lookup` field. The `!` option will navigate the user back to the reference field to add/modify/delete reference field entries. The `v` option will show the user all available values that can be used in the currently displyed `lookup` field.

The following example shows configuration of node-groups-based fields via the configurator user interface. In this example, the login nodes are configured using `lookup` fields in the `cray_login` configurator template. After the new configurator templates have been installed on the system, begin by running `cfgset update` on the config set to be updated. This example uses the `-s cray_login` option to specifically target and update only the `cray_login` service within a config set p0.

```
smw# cfgset update -s cray_login p0
INFO - Running pre-configuration scripts
INFO - Validating templates and configuration data. One moment please.
INFO - Validating configuration templates for YAML syntax.
INFO - Validating configuration templates for schema compliance.
INFO - Template 'cray_image_layering_config.yaml' found in config set with no
matching template to merge.
INFO - Merging configuration templates and validating schema.
INFO - Validating configuration templates for lookup resolution.
************ cray_login.settings.login_nodes.data.member_groups ************
```

```
      member_groups -- Groups of login nodes
        A list of all internal node groups that will be used as login or mom
        nodes. The nodes in these groups are nodes which have been connected
        via ethernet to the system and are to be used by end users to access
        the system.

        Enter the node groups of the login nodes.

        Default:           Current:
            (none)         (none)

    Value: lookup, blank values not allowed
           level=basic, state=unset

    Inputs: menu commands (? for help)


|--- Information
|  *    The values for this field need to be defined in the following setting:
|  *      - 'cray_node_groups.settings.groups'
|  *
|  *    Type '!' at the prompt to modify 'cray_node_groups.settings.groups'
|  *    Type 'v' at the prompt to view values of 'cray_node_groups.settings.groups'
|---

cray_login.settings.login_nodes.data.member_groups
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $
```

The configurator presents a new field called `member_groups` that needs to be configured. The `member_groups` setting is a `lookup` field, meaning it will expect whatever values the user enters to be present in the field it references. In this case, the reference field is `cray_node_groups.settings.groups`, the field that defines node groups. To view acceptable values for a `lookup` field, type `v` at the prompt. In this example, typing `v` will display the values currently defined in the `cray_node_groups.settings.groups` setting. The output below shows the result of typing `v` and pressing `Enter` at the prompt.

```
********** cray_login.settings.login_nodes.data.member_groups **********
      member_groups -- Groups of login nodes
        A list of all internal node groups that will be used as login or mom
        nodes. The nodes in these groups are nodes which have been connected
        via ethernet to the system and are to be used by end users to access
        the system.
        Enter the node groups of the login nodes.
        Default:           Current:
            (none)         (none)
    Value: lookup, blank values not allowed
           level=basic, state=unset
    Inputs: menu commands (? for help)
|--- Information
|  *    The values for this field need to be defined in the following setting:
|  *      - 'cray_node_groups.settings.groups'
|  *
|  *    Type '!' at the prompt to modify 'cray_node_groups.settings.groups'
|  *    Type 'v' at the prompt to view values of 'cray_node_groups.settings.groups'
|  *    The values of the reference field 'cray_node_groups.settings.groups' are:
|  *      - compute_nodes
|  *      - service_nodes
|  *      - smw_nodes
|  *      - boot_nodes
```

```
|  *      - sdb_nodes
|  *      - login_nodes
|  *      - all_nodes
|  *      - tier2_nodes
|---
cray_login.settings.login_nodes.data.member_groups
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $
```

The configurator allows the user to temporarily switch to the `cray_node_groups.settings.groups` setting to define a new node group if none of the existing node groups are appropriate for the current `lookup` field being configured. Type `!` at the prompt to temporarily switch to the refence field to add, delete, or modify reference field entires. The output below shows the result of typing `!` and pressing `Enter`:

```
****************** cray_node_groups.settings.groups ******************
      groups
        Define node groups for referencing lists of CLE nodes in other CLE
        configuration data fields in this config set. Node groups can be
        arbitrary groupings of CLE nodes, but nodes within a group are often
        related by specific software functionality or hardware characteristics.
        Examples of commonly-defined groups are login nodes, DVS servers, RSIP
        servers, etc. Each group may contain multiple nodes, and nodes may be
        included in more than one node group. Nodes do not need to be
        explicitly assigned to a node group.
    Configured Values:
        1) 'compute_nodes'
            a) members:
                   platform:compute
        2) 'service_nodes'
            a) members:
                   platform:service
         ... 6 more groups entries...
    Inputs: menu commands (? for help)
|--- Information
|  *    You were editing 'cray_login.settings.login_nodes.data.member_groups'
|  *    The configurator will return to that field when you are finished here.
|  *
|  *    Multiple 'groups' entries can be added using this menu
|---
cray_node_groups.settings.groups
[<cr>=set 8 entries, +=add an entry, ?=help, @=less] $
```

At this point, the configurator is editing the `groups` setting of the `cray_node_groups` configurator service. The output below shows how to edit the login nodes group to add the cname of the login node to be added to the group. First, show all defined groups by typing `*` and pressing `Enter`:

```
****************** cray_node_groups.settings.groups ******************
      groups
        Define node groups for referencing lists of CLE nodes in other CLE
        configuration data fields in this config set. Node groups can be
        arbitrary groupings of CLE nodes, but nodes within a group are often
        related by specific software functionality or hardware characteristics.
        Examples of commonly-defined groups are login nodes, DVS servers, RSIP
        servers, etc. Each group may contain multiple nodes, and nodes may be
        included in more than one node group. Nodes do not need to be
        explicitly assigned to a node group.
    Configured Values:
        1) 'compute_nodes'
            a) members:
                   platform:compute
```

```
       2) 'service_nodes'
           a) members:
                  platform:service
       3) 'smw_nodes'
           a) members: (none)
       4) 'boot_nodes'
           a) members: (none)
       5) 'sdb_nodes'
           a) members: (none)
       6) 'login_nodes'
           a) members: (none)
       7) 'all_nodes'
           a) members:
                  platform:compute
                  platform:service
       8) 'tier2_nodes'
           a) members: (none)
    Inputs: menu commands (? for help)
|--- Information
|  *    No viewing limit set. Type '*' to limit view to 2 entries.
|---
cray_node_groups.settings.groups
[<cr>=set 8 entries, +=add an entry, ?=help, @=less] $
```

Above, the configurator shows a node group defined with the name `login_nodes`, so this example will add login nodes to the members list of that group. Modify that group's list of members by typing the number and letter of the field to edit followed by an asterisk. In this case, entering `6a*` will select `login_nodes → members`. Press `Enter` to confirm selection:

```
********* cray_node_groups.settings.groups.data.login_nodes.members **********
    groups (current key: login_nodes)
         Define node groups for referencing lists of CLE nodes in other CLE
         configuration data fields in this config set. Node groups can be
         arbitrary groupings of CLE nodes, but nodes within a group are often
         related by specific software functionality or hardware
         characteristics. Examples of commonly-defined groups are login nodes,
         DVS servers, RSIP servers, etc. Each group may contain multiple nodes,
         and nodes may be included in more than one node group. Nodes do not
         need to be explicitly assigned to a node group.
      members -- Node Group Member List
          Supply the name of the nodes that are members of this group. For CLE
          nodes, this is the c-name of the node. If an SMW node is included in
          the group, enter the output of the 'hostid' command from the SMW
          node. For eLogin nodes, the output of the 'hostname' command should
          be used. The 'platform:compute' and 'platform:service' keywords can
          be used to include all compute or service nodes in the current
          partition in the group, respectively. Group members prefixed by a
          tilde (~) are excluded from the group. This applies to c-names,
          hostids, hostnames, and the 'platform:service' or 'platform:compute'
          keywords.
        Default:           Current:
           (none)          (none)
    Value: list, blank values not allowed, regex=^~?c\d+-\d+c[0-2]s(\d|
1[0-5])n[0-3]$|^~?platform:service$|^~?platform:compute$|^~?[0-9a-f]{8}|^~?[A-Za-
z0-9][A-Za-z0-9-]{0,252}$
          level=required, state=all
    Inputs: menu commands (? for help)
cray_node_groups.settings.groups.data.login_nodes.members
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $
```

Type + to add desired login nodes' cnames to this node group. Then type the login node cnames, one per line, and terminate the list of members by pressing `Ctrl-d` (EOF):

```
********* cray_node_groups.settings.groups.data.login_nodes.members **********
    groups (current key: login_nodes)
         Define node groups for referencing lists of CLE nodes in other CLE
         configuration data fields in this config set. Node groups can be
         arbitrary groupings of CLE nodes, but nodes within a group are often
         related by specific software functionality or hardware
         characteristics. Examples of commonly-defined groups are login nodes,
         DVS servers, RSIP servers, etc. Each group may contain multiple nodes,
         and nodes may be included in more than one node group. Nodes do not
         need to be explicitly assigned to a node group.
      members -- Node Group Member List
          Supply the name of the nodes that are members of this group. For CLE
          nodes, this is the c-name of the node. If an SMW node is included in
          the group, enter the output of the 'hostid' command from the SMW
          node. For eLogin nodes, the output of the 'hostname' command should
          be used. The 'platform:compute' and 'platform:service' keywords can
          be used to include all compute or service nodes in the current
          partition in the group, respectively. Group members prefixed by a
          tilde (~) are excluded from the group. This applies to c-names,
          hostids, hostnames, and the 'platform:service' or 'platform:compute'
          keywords.
        Default:           Current:
          (none)           (none)
    Value: list, blank values not allowed, regex=^~?c\d+-\d+c[0-2]s(\d|
1[0-5])n[0-3]$|^~?platform:service$|^~?platform:compute$|^~?[0-9a-f]{8}|^~?[A-Za-
z0-9][A-Za-z0-9-]{0,252}$
          level=required, state=all
    Inputs: menu commands (? for help)
cray_node_groups.settings.groups.data.login_nodes.members
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $ +
Add members (Ctrl-d to exit) $ c0-0c0s0n2
Add members (Ctrl-d to exit) $ c0-0c0s1n1
Add members (Ctrl-d to exit) $
```

The configurator then displays the list of nodes entered for the `members` field of the `login_nodes` node group:

```
********** cray_node_groups.settings.groups.data.login_nodes.members **********
    groups (current key: login_nodes)
         Define node groups for referencing lists of CLE nodes in other CLE
         configuration data fields in this config set. Node groups can be
         arbitrary groupings of CLE nodes, but nodes within a group are often
         related by specific software functionality or hardware
         characteristics. Examples of commonly-defined groups are login nodes,
         DVS servers, RSIP servers, etc. Each group may contain multiple nodes,
         and nodes may be included in more than one node group. Nodes do not
         need to be explicitly assigned to a node group.
      members -- Node Group Member List
          Supply the name of the nodes that are members of this group. For CLE
          nodes, this is the c-name of the node. If an SMW node is included in
          the group, enter the output of the 'hostid' command from the SMW
          node. For eLogin nodes, the output of the 'hostname' command should
          be used. The 'platform:compute' and 'platform:service' keywords can
          be used to include all compute or service nodes in the current
          partition in the group, respectively. Group members prefixed by a
          tilde (~) are excluded from the group. This applies to c-names,
          hostids, hostnames, and the 'platform:service' or 'platform:compute'
          keywords.
```

```
        Default:           Current:
            (none)             1) c0-0c0s0n2
                               2) c0-0c0s1n1
    Value: list, blank values not allowed, regex=^~?c\d+-\d+c[0-2]s(\d|
1[0-5])n[0-3]$|^~?platform:service$|^~?platform:compute$|^~?[0-9a-f]{8}|^~?[A-Za-
z0-9][A-Za-z0-9-]{0,252}$
           level=required, state=all
    Inputs: menu commands (? for help)
|--- Information
|  *    2 entries modified. Press <cr> to set.
|---
cray_node_groups.settings.groups.data.login_nodes.members
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $
```

Press `Enter` to accept the values. This brings the user back to the `groups` setting:

```
******************** cray_node_groups.settings.groups ********************
      groups
        Define node groups for referencing lists of CLE nodes in other CLE
        configuration data fields in this config set. Node groups can be
        arbitrary groupings of CLE nodes, but nodes within a group are often
        related by specific software functionality or hardware characteristics.
        Examples of commonly-defined groups are login nodes, DVS servers, RSIP
        servers, etc. Each group may contain multiple nodes, and nodes may be
        included in more than one node group. Nodes do not need to be
        explicitly assigned to a node group.
    Configured Values:
        1) 'compute_nodes'
            a) members:
                   platform:compute
        2) 'service_nodes'
            a) members:
                   platform:service
        3) 'smw_nodes'
            a) members: (none)
        4) 'boot_nodes'
            a) members: (none)
        5) 'sdb_nodes'
            a) members: (none)
        6) 'login_nodes'
            a) members:
                   c0-0c0s0n2
                   c0-0c0s1n1
        7) 'all_nodes'
            a) members:
                   platform:compute
                   platform:service
        8) 'tier2_nodes'
            a) members: (none)
    Inputs: menu commands (? for help)
|--- Information
|  *    Multiple 'groups' entries can be added using this menu
|---
cray_node_groups.settings.groups
[<cr>=set 8 entries, +=add an entry, ?=help, @=less] $
```

At this point, more node groups may be defined by typing + and pressing `Enter`. Adding additional node groups works just like adding elements to other `multival` type settings in the configurator. In this example, all desired `login_nodes` node group members have been entered. Press `Enter` to leave

`cray_node_groups.settings.groups` and return to `cray_login.settings.login_nodes.data.member_groups`:

```
********** cray_login.settings.login_nodes.data.member_groups **********
      member_groups -- Groups of login nodes
        A list of all internal node groups that will be used as login or mom
        nodes. The nodes in these groups are nodes which have been connected
        via ethernet to the system and are to be used by end users to access
        the system.
        Enter the node groups of the login nodes.
        Default:           Current:
            (none)         (none)
    Value: lookup, blank values not allowed
           level=basic, state=all
    Inputs: menu commands (? for help)
|--- Information
|  *    The values for this field need to be defined in the following setting:
|  *      - 'cray_node_groups.settings.groups'
|  *
|  *    Type '!' at the prompt to modify 'cray_node_groups.settings.groups'
|  *    Type 'v' at the prompt to view values of 'cray_node_groups.settings.groups'
|---
cray_login.settings.login_nodes.data.member_groups
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $
```

Now that the desired login nodes have been added to the node group named `login_nodes` as defined in the reference field `cray_node_groups.settings.groups`, the `login_nodes` group can be added to the `cray_login.settings.login_nodes.data.member_groups` setting by typing `+`, pressing `Enter`, and entering the name of the node group:

```
********** cray_login.settings.login_nodes.data.member_groups **********
      member_groups -- Groups of login nodes
        A list of all internal node groups that will be used as login or mom
        nodes. The nodes in these groups are nodes which have been connected
        via ethernet to the system and are to be used by end users to access
        the system.
        Enter the node groups of the login nodes.
        Default:           Current:
            (none)         (none)
    Value: lookup, blank values not allowed
           level=basic, state=all
    Inputs: menu commands (? for help)
|--- Information
|  *    The values for this field need to be defined in the following setting:
|  *      - 'cray_node_groups.settings.groups'
|  *
|  *    Type '!' at the prompt to modify 'cray_node_groups.settings.groups'
|  *    Type 'v' at the prompt to view values of 'cray_node_groups.settings.groups'
|---
cray_login.settings.login_nodes.data.member_groups
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $ +
Add member_groups (Ctrl-d to exit) $ login_nodes
Add member_groups (Ctrl-d to exit) $
```

Pressing `Ctrl-d` terminates the list and displays the value of the `member_groups` field:

```
********** cray_login.settings.login_nodes.data.member_groups **********
      member_groups -- Groups of login nodes
        A list of all internal node groups that will be used as login or mom
```

```
        nodes. The nodes in these groups are nodes which have been connected
        via ethernet to the system and are to be used by end users to access
        the system.
        Enter the node groups of the login nodes.
        Default:           Current:
            (none)             1) login_nodes
    Value: lookup, blank values not allowed
           level=basic, state=all
    Inputs: menu commands (? for help)
|--- Information
|  *    1 entry modified. Press <cr> to set.
|---
cray_login.settings.login_nodes.data.member_groups
[<cr>=set 1 entries, +=add an entry, ?=help, @=less] $
```

Pressing `Enter` again finishes configuring all new settings that need to be configured for node groups. Once the user saves and exits the configurator, the `member_groups` setting of the `cray_login` service will be configured with the node group that is configured in the `cray_node_groups` service as shown below in the output of cfgset search of a config set p0:

```
smw# cfgset search -s cray_login --level basic p0

# 2 matches for '.' from cray_login_config.yaml
#-------------------------------------------------------------------------------
cray_login.settings.login_nodes.data.member_groups: login_nodes
cray_login.settings.login_nodes.data.login_prohibited_after_boot: false

smw# cfgset search -s cray_node_groups --level basic p0

# 16 matches for '.' from cray_node_groups_config.yaml
#-------------------------------------------------------------------------------
cray_node_groups.settings.groups.data.compute_nodes.description: Default node group
which contains all the compute nodes for the current partition.
cray_node_groups.settings.groups.data.compute_nodes.members: platform:compute
cray_node_groups.settings.groups.data.service_nodes.description: Default node group
which contains all the service nodes for the current partition.
cray_node_groups.settings.groups.data.service_nodes.members: platform:service
cray_node_groups.settings.groups.data.smw_nodes.description: Default node group
which contains the primary and failover (if applicable) SMW nodes.
cray_node_groups.settings.groups.data.smw_nodes.members: [ ] # (empty)
cray_node_groups.settings.groups.data.boot_nodes.description: Default node group
which contains the primary and failover (if applicable) boot nodes associated with
the current partition.
cray_node_groups.settings.groups.data.boot_nodes.members: [ ] # (empty)
cray_node_groups.settings.groups.data.sdb_nodes.description: Default node group
which contains the primary and failover (if applicable) SDB nodes associated with
the current partition.
cray_node_groups.settings.groups.data.sdb_nodes.members: [ ] # (empty)
cray_node_groups.settings.groups.data.login_nodes.description: Default node group
which contains the login nodes for the configured system.
cray_node_groups.settings.groups.data.login_nodes.members: c0-0c0s0n2, c0-0c0s1n1
cray_node_groups.settings.groups.data.all_nodes.description: Default node group
which contains all of the nodes applicable to the current system. May also contain
SMW nodes and external login nodes.
cray_node_groups.settings.groups.data.all_nodes.members: platform:compute,
platform:service
cray_node_groups.settings.groups.data.tier2_nodes.description: Default node group
which contains the tier 2 nodes in the system. See the guidance in the
cray_scalable_services service for a detailed description of tier 2 nodes.
cray_node_groups.settings.groups.data.tier2_nodes.members: [ ] # (empty)
```

4.3 Configurator Screens and Menus

Service Configuration List Menu (Interactive Mode)

When the configurator is invoked in interactive mode for all services, it presents the **Service Configuration List Menu** (also referred to as Service List Menu), which displays all services currently in the config set that meet the specified state and level filters.

Here is an example command and the resulting menu for a config set `p0.staging` of type `cle`. The list includes only services with configuration level `basic` and state `unset` (not included in command because those are defaults).

```
smw# cfgset update --mode interactive p0.staging
```

```
Service Configuration List Menu (Config Set: p0.staging, type: cle)

-------------------------------------------------------------------------------
  Selected     #      Service           Status (level=basic, state=unset)
-------------------------------------------------------------------------------
               1)     cray_example      unconfigured service, 8/8 unconfigured
settings
               2)     cray_example2     [ OK ], 6/7 unconfigured settings
-------------------------------------------------------------------------------

**** Select Options ****
  a: all                       n: none                      c: configured
  u: unconfigured              e: enabled                   d: disabled
  i: inheriting                #: toggle #

**** Actions on Selected (0 services) ****
  C: configure                 E: toggle enable             I: toggle inherit
  @: show guidance             v: view settings

**** Other Actions ****
  ?: help                      l: switch level              s: switch state
  r: refresh                   $: view changelog            Q: save & exit
  x: exit without save

Service List Menu [default: save & exit - Q] $
```

Information about the config set is shown at the top after the menu title. The services are presented in a list, followed by options for operations on the services. Multiple services can be selected using the filters listed in the **Select Options** submenu. For example, to select all of the unconfigured services in the list, enter **u** at the prompt (in this case, the `cray_example` service would be selected). Services can be selected by entering a range of numbers (e.g., **`1-2`**), a list of numbers and ranges (e.g., **`1-2,4,7-9`**), or by number directly (e.g., entering **2** will select the `cray_example2` service). To deselect all selections, enter **n**, and to deselect one or a subset of services, select them again to toggle selection off.

After one or more services are selected, the **Actions On Selected** submenu can be used to perform operations on the selected services. The following options are available:

C Configures the services by entering auto mode and presenting configuration setting screen(s), then returns to interactive mode and this menu when complete.

E Switches each selected service to enabled if currently disabled and vice versa.

I Switches each selected service to inheriting if not and vice versa. Applies only to services in the current config set that have a global version of the configuration template.

@ Shows the top-level service guidance for each service.

v View the service menu for the selected service (this option appears only when a single service is selected, not when multiple are selected).

When one or more services are selected, the default menu option changes from **save & exit - Q** to **configure - C**. Pressing **Enter** when one or more services are selected will begin configuration on the selected services.

The **Other Actions** submenu provides options that act on all services or on the current configurator session. The following options are available:

? Display help content on all input options in this menu and short descriptions of the commands.

l Switch the configuration level from the current level to the next level. The level is cycled from `basic` to `advanced` to `required` to `basic` and so on. The menu will update with the effects of the filtering.

s Switch the configuration state from the current state to the next state. The state is cycled from `unset` to `set` to `all` to `unset` and so on. The menu will update with the effects of the filtering.

r Refresh the screen by clearing any help, guidance, and information messages.

$ View a list of changes that have been made to the services in the current session.

Q Exit the configuration session completely and save any changes.

x Exit the configuration session completely and do not save any changes.

Service Configuration Menu (Interactive Mode)

When the configurator is invoked in interactive mode for a specific service, it presents the **Service Configuration Menu** (service menu), which displays the specified service. The configurator also displays this menu if the **v** (view service settings) option is used when that service is selected in the Service List Menu.

Here is an example command and the resulting menu for service`cray_example` in config set `p0.staging` of type `cle`. The list of settings includes only those with configuration level `basic` (not included in command because it is the default). In the service menu, settings of all states are shown, regardless of any state specified on the command line.

```
smw# cfgset update --mode interactive --service cray_example p0.staging

Service Configuration Menu (Config Set: p0.staging, type: cle)
  cray_example        [ status: enabled ]  [ validation: valid ]

------------------------------------------------------------------------
  Selected   #     Settings                  Value/Status (level=basic)
------------------------------------------------------------------------
                   basic_example
             1)      basic_string_value       [ unconfigured, default=abc ]
             2)      basic_protected_value    [ unconfigured, default=(none) ]
             3)      basic_list_value         [ unconfigured, default=(none) ]
             4)    multival_example           [ 4 sub-settings unconfigured,
                                                select and enter C to add entries ]
------------------------------------------------------------------------
**** Select Options ****
```

```
  a: all                        n: none                       c: configured
  u: unconfigured               #: toggle #

**** Actions on Selected (0 settings) ****
  C: configure                  @: show guidance

**** Other Actions ****
  ?: help                       l: switch level               E: toggle enable
  I: toggle inherit             ^^: go to service list        r: refresh
  $: view changelog             Q: save & exit                x: exit without save

Cray Example Service Menu [default: save & exit - Q] $
```

> **NOTE:** A status of `validation: valid` as seen in the above screenshot does not take into account validation rules.

Information about the config set and the status of the service is shown at the top of the menu. The settings within the service are presented in a list, followed by options for operations on the settings. Multiple settings can be selected using the filters listed in the **Select Options** submenu. For example, to select all of the unconfigured settings in the list, enter **u** at the prompt (in this case, all settings would be selected). Settings can be selected by entering a range of numbers (e.g., **`1-2`**), a list of numbers and ranges (e.g., **`1-2,4,7-9`**), or by number directly (e.g., entering **2** will select the `basic_protected_value` setting). To deselect all selections, enter **n**, and to deselect one or a subset of settings, select them again to toggle selection off.

After one or more settings are selected, the **Actions On Selected** submenu can be used to perform operations on the selected settings and fields. The following options are available:

C Configures the settings by entering auto mode and presenting configuration setting screen(s), then returns to interactive mode and this menu when complete.

@ Shows the guidance for each field.

When one or more settings are selected, the default menu option changes from **save & exit - Q** to **configure - C**. Pressing **Enter** when one or more services are selected will display the configuration setting screen so that the user can begin configuration on the selected settings. See *Configuration Setting Screen (Interactive and Auto Mode)*.

The **Other Actions** submenu provides options that act on all settings or on the current configurator session. The following options are available:

? Display help content on all input options in this menu and short descriptions of the commands.

l Switch the configuration level from the current level to the next level. The level is cycled from `basic` to `advanced` to `required` to `basic` and so on. The menu will update with the effects of the filtering.

E Switches the service to enabled if disabled and vice versa.

I Switches the service to inheriting if not and vice versa. Applies only to services in the current config set that have a global version of the configuration template.

^^ Navigate up to the Service List Menu (effective only if the configurator was invoked on all services, i.e., without the `--service` option).

r Refresh the screen by clearing any help, guidance, and information messages.

$ View a list of changes that have been made to the service in the current session.

Q Exit the configuration session completely and save any changes.

x Exit the configuration session completely and do not save any changes.

Configuration Setting Screen (Interactive and Auto Mode)

When the configurator is in auto mode, it presents a configuration setting screen, which displays a setting to be configured. The configurator can get into auto mode by being invoked in that mode, or by dropping into it from interactive mode, when the user drills down to configure a particular setting/field.

Here is the template for a configuration setting screen.

```
************************* {{path.to.field}} *************************
      {{field name} -- {{field title}}
        {{ field guidance }}

        Default:                Current:
            {{default value}}       {{current value, if configured}}

    Value: {{data type}}, {{validation criteria}}
           {{configuration level/state}}

    Inputs: <{{data type}}>  -- OR --  menu commands (? for help)

|--- Information/Errors
|  *    {{messages}}
|---

[<cr>=set '{{default value}}', <new value>, ?=help, @={{more/less}}] $
```

Here is an example of invoking the configurator in auto mode for a config set `p0.staging` of type `cle`. The configurator will present only settings with configuration level `basic` and state `unset` (not included in command because those are defaults).

```
smw# cfgset create p0.staging
```

This is the configuration setting screen for the field `basic_string_value` in the `cray_example` service in config set `p0.staging`.

```
******* cray_example.settings.basic_example.data.basic_string_value ******
      basic_string_value -- Basic String Value
        This field is part of the basic_example setting in the cray_example
        service. It is a string value that only allows lowercase letters. The
        configurator will validate the value for this field that it is a string
        and matches the letters. Its default value is 'abc'.

        Default:       Current:
            abc            not configured yet

     Value: string, blank values not allowed, regex=^[a-z]*$
            level=basic, state=unset

     Inputs: <string> -- OR --  menu commands (? for help)

cray_example.settings.basic_example.data.basic_string_value
[<cr>=set 'abc', <new value>, ?=help, @=less] $
```

The first line of output is the full name of the setting field (or 'path.to.field' because it is the exact location of the data value for the field in the config set configuration template). The full name is output in `cfgset search` output, and it is also used in configuration worksheets.

The last line of output is a prompt to accept input. Information messages are shown only when requested through a menu command, and error messages are shown only when incorrect input is received. The following options are available:

<cr> (Enter or Return key)	User presses **Enter** to accept the default value, which in this case is setting `basic_string_example` to the value `abc`.
<new value>	User enters a new value at the prompt and then presses **Enter** to to change `basic_string_example` from its current or default value.
?	User enters **?** at the prompt and then presses **Enter** to print the contextual help menu to the screen.
@	User enters @ at the prompt and then presses **Enter** to toggle configurator verbosity to show more or less information about the `basic_string_example` field.

Navigating in a Configuration Setting Screen (Interactive and Auto Mode)

When navigating through the configuration settings for a given service, it may be necessary to go back and review a previous setting or skip a current setting so it can be configured at a later time. The configurator user interface enables the user to do this with a set of commands that are entered at the prompt in the configuration setting screen. One of these commands enables users to switch to interactive mode by navigating up to the service menu and Service List Menu. The following navigation options are available:

> Skip to the next configuration setting field screen. The configurator will not mark the current screen's field as being configured if this command is selected.

< Navigate back to the previous setting field screen. If the current screen is the first field that was marked for configuration, use the commands to navigate to the service menu or Service List Menu instead.

^ Navigate "up" from the configuration setting screen to the service menu. This places the configurator in interactive mode.

^^ Navigate "up" from the configuration setting screen or the service menu to the Service List Menu (this navigates to the service menu if the `--service` option was added when invoking the configurator originally). This places the configurator in interactive mode.

4.4 Basic Configurator UI Operations

Change Configuration Filters During a Session

To change configuration filters during a configurator session when in interactive mode:

- Enter **l** (lowercase L) at the menu prompt to cycle through the level filters.
- Enter **s** at the menu prompt to cycle through the state filters. Applicable only in the Service List Menu; all states are displayed in a service menu.

Switch Configuration Modes During a Session

To switch to interactive mode when in a configuration setting screen (displayed when configurator invoked in auto mode or when configuring selected services/settings in interactive mode):

- Enter **^** at the menu prompt to display the service menu.
- Enter **^^** at the menu prompt to display the Service List Menu. Applicable only if the configurator was invoked without the `--service` option.

View Session Changes

To view all changes to configuration data made (so far) during a configurator session when in interactive mode (in either the Service List Menu or a service menu):

- Enter **$** at the menu prompt to display setting and field name(s) along with previous and current values if anything has changed during the configurator session.

These changes are also provided in the `changelog` subdirectory of the config set for each completed configurator session.

Exit a Configurator Session (with or without saving changes)

To exit a configurator session from any configurator screen or menu, whether in auto or interactive mode:

- Enter **Q** at the menu prompt to exit and save all changes.
- Enter **x** at the menu prompt to exit and without saving changes.

  IMPORTANT: Because configuration callback scripts are run before and after a configurator session, the config set may not be exactly the same even if the configurator session is exited without saving changes. Exiting without saving changes only guarantees that any changes to configuration data made *during the session* are not saved.

5 Common Tasks When Using the Configurator Interactively

The following tasks are easy to do using the configurator in interactive mode. All assume that a config set already exists, and all use configuration settings in service `cray_example` in config set `p0.staging` to illustrate how to perform the tasks. Most also assume that the name of the configuration setting is known as well as which service(s) contain it. That information is necessary to invoke the configurator to update only the necessary service and setting. If that information is not known, see *Locate a Configuration Parameter in a Config Set* on page 64.

- *Change a Basic Setting Field during a Configurator Session* on page 65, where a basic setting is of type string, integer, float, or boolean.
- *Change a List Setting Field during a Configurator Session* on page 67.
- *Change a Multival Setting Field during a Configurator Session* on page 69, which is a setting with one or more subsettings (requires an extra step or two to edit).
- *Change the Service Enabled/Disabled Status during a Configurator Session* on page 72.
- *Change Service Inheritance during a Configurator Session* on page 73, which applies only to those services with templates of both types: `cle` and `global`.
- *Revert a Field to its Default Value during a Configurator Session* on page 74.

5.1 Locate a Configuration Parameter in a Config Set

Here are suggestions for finding out whether a parameter is in a config set and if so, which service(s) it belongs to.

Use `cfgset search`

If the setting or field corresponding to the parameter of interest is known, use the `search` subcommand. For example, to locate the field `basic_string_value` in config set `p0.staging`:

```
smw# cfgset search --term basic_string_value --level advanced p0.staging
```

The configurator displays the following results.

```
# 3 matches for 'basic_list_value' from cray_example_config.yaml
#-------------------------------------------------------------------------------
cray_example.settings.basic_example.data.basic_string_value: abc
cray_example.settings.multival_example.data.foo.basic_string_value: foovalue
cray_example.settings.multival_example.data.bar.basic_string_value: barvalue

# 1 match for 'basic_list_value' from cray_example2_config.yaml
#-------------------------------------------------------------------------------
cray_example2.settings.basic_example.data.basic_string_value: def
```

In this example, assume that the particular field that needs to be changed is `cray_example.settings.basic_example.data.basic_string_value`. This field is found in the `cray_example` service, `basic_example` setting. To change that field, see *Change a Basic Setting Field during a Configurator Session* on page 65.

Navigate through interactive menus

Navigating through the interactive menus can be helpful when the user has some idea of what they are looking for and can narrow it down to a single service or two. For example, to see all services in config set `p0.staging`:

```
smw# cfgset update --mode interactive --level advanced p0.staging
```

Print out all settings in the config set

To print out the entire config set, simply search the config set and omit the `--term` option.

For example, to print out all required settings in config set `p0.staging` that are not yet configured:

```
smw# cfgset search --level required --state unset p0.staging
```

To print out all required and basic settings, both set and unset (uses defaults for level and state):

```
smw# cfgset search p0.staging
```

To print out all settings (uses default for state):

```
smw# cfgset search --level advanced p0.staging
```

5.2 Change a Basic Setting Field during a Configurator Session

Prerequisites

This procedure assumes that the config set to be updated already exists and the name and location (service) of the setting field to be changed is known. If the name or location is not known, see *Locate a Configuration Parameter in a Config Set* on page 64.

About this task

This procedure describes how to change a basic setting field (i.e., a field of type string, integer, float, or boolean) using the example of changing field `basic_string_value` in service `cray_example` in config set `p0.staging`.

Procedure

1. Invoke the configurator in interactive mode for the service that contains the setting field.

   ```
   smw# cfgset update --mode interactive --level advanced --service cray_example
   p0.staging
   ```

   The configurator displays the Service Configuration Menu (service menu). The field to be changed is #1 in the list of settings.

```
Service Configuration Menu (Config Set: p0.staging, type: cle)

  cray_example        [ status: enabled ]  [ validation: valid ]

--------------------------------------------------------------------------------
  Selected     #      Settings                    Value/Status (level=advanced)
--------------------------------------------------------------------------------
                      basic_example
               1)       basic_string_value       abc
               2)       basic_protected_value    *** <hidden> ***
               3)       basic_list_value         first_value, second_value
               4)       basic_integer_value      456

               5)     multival_example
                         hostname: foo           [ OK ]
                         hostname: bar           [ OK ]
--------------------------------------------------------------------------------
**** Select Options ****
  a: all                    n: none                    c: configured
  u: unconfigured           #: toggle #

**** Actions on Selected (1 settings) ****
  C: configure              @: show guidance

**** Other Actions ****
  ?: help                   l: switch level            E: toggle enable
  I: toggle inherit         ^^: go to service list     r: refresh
  $: view changelog         Q: save & exit             x: exit without save

Cray Example Service Menu [default: save & exit - Q] $
```

2. Select the setting/field to be changed.

```
Cray Example Service Menu [default: save & exit - Q] $ 1
```

The configurator redraws the service menu with an asterisk next to the selected item and a different prompt.

3. Configure the selected setting/field.

Enter **C** or press **Enter** (<cr>) to accept the default action of `configure - C`.

```
Cray Example Service Menu [default: configure - C] $ C
```

The configurator displays the configuration setting screen for the `basic_string_value` field.

```
****** cray_example.settings.basic_example.data.basic_string_value *******

      basic_string_value -- Basic String Value
        This field is part of the basic_example setting in the cray_example
        service. It is a string value that only allows lowercase letters. The
        configurator will validate the value for this field that it is a string
        and matches the letters only validation criteria. Its default value is
        'abc'.

        Default:          Current:
            abc               abc

    Value: string, blank values not allowed, regex=^[a-z]*$
           level=basic, state=unset
```

```
    Inputs: <string>  -- OR --  menu commands (? for help)

cray_example.settings.basic_example.data.basic_string_value
[<cr>=keep 'abc', <new value>, ?=help, @=less] $
```

The user can now make the desired changes to this field.

4. Enter the new value, then press **Enter** to set the value of that field.

```
cray_example.settings.basic_example.data.basic_string_value
[<cr>=keep 'abc', <new value>, ?=help, @=less] $ hello
```

5.3 Change a List Setting Field during a Configurator Session

Prerequisites

This procedure assumes that the config set to be updated already exists and the name and location (service) of the setting field to be changed is known. If the name or location is not known, see *Locate a Configuration Parameter in a Config Set* on page 64.

About this task

This procedure describes how to change a list field using the example of changing field `basic_list_value` in service `cray_example` in config set `p0.staging`.

Procedure

1. Invoke the configurator in interactive mode for the service that contains the setting field.

```
smw# cfgset update --mode interactive --level advanced --service cray_example
p0.staging
```

The configurator displays the Service Configuration Menu (service menu). The field to be changed is #3 in the list of settings.

```
Service Configuration Menu (Config Set: p0.staging, type: cle)

  cray_example        [ status: enabled ]  [ validation: valid ]

--------------------------------------------------------------------------------
  Selected     #      Settings                    Value/Status (level=advanced)
--------------------------------------------------------------------------------
                      basic_example
               1)       basic_string_value       abc
               2)       basic_protected_value    *** <hidden> ***
               3)       basic_list_value         first_value, second_value
               4)       basic_integer_value      456

               5)     multival_example
                        hostname: foo            [ OK ]
                        hostname: bar            [ OK ]
--------------------------------------------------------------------------------
**** Select Options ****
  a: all                     n: none                     c: configured
```

```
  u: unconfigured             #: toggle #

**** Actions on Selected (1 settings) ****
  C: configure                @: show guidance

**** Other Actions ****
  ?: help                     l: switch level             E: toggle enable
  I: toggle inherit           ^^: go to service list      r: refresh
  $: view changelog           Q: save & exit              x: exit without save

Cray Example Service Menu [default: save & exit - Q] $
```

2. Select the setting/field to be changed.

```
Cray Example Service Menu [default: save & exit - Q] $ 3
```

The configurator redraws the service menu with an asterisk next to the selected item and a different prompt.

3. Configure the selected setting.

Enter **C** or press **Enter** (<cr>) to accept the default action of `configure - C`.

```
Cray Example Service Menu [default: configure - C] $ C
```

The configurator displays the configuration setting screen for the `basic_list_value` field.

```
******* cray_example.settings.basic_example.data.basic_list_value *******

      basic_list_value -- Basic List Value
        This field is part of the basic_example setting in the cray_example
        service. It is a list value. The input mechanism is slightly different
        for protected values.

        Default:           Current:
            (none)             1) first_value
                               2) second_value

    Value: list, blank values not allowed
           level=basic, state=unset

    Inputs: menu commands (? for help)

|--- Command Help
|  *    +     - add entries
|  *    <#>*  - change the <#> entry. '2*' changes entry 2
|  *    <#>-  - delete the <#> entry. '4-' deletes entry 4
|  *    d     - delete all entries in the list
|  *    u     - undo all changes
|  *    <cr>  - accept the current value(s)
|  *    #     - set the value to its default
|  *       [... some help entries removed ...]
|  *    ?     - show this help
|
| Press Enter/return to take default action (current: <cr> - accept), OR enter a
new value at prompt
|---

cray_example.settings.basic_example.data.basic_list_value
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $
```

The user can now make the desired changes to this field. In this example, assume that the second entry in `basic_list_value` needs to be changed.

4. Select which list entry to change and enter the new value.

   Select the second entry, enter a new value, then press **Enter**. Use the **(Ctrl-d to exit)** instruction only to cancel the modification, if necessary.

```
cray_example.settings.basic_example.data.basic_list_value
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ 2*
Modify basic_list_value second_value (Ctrl-d to exit) $ second_value_new
```

   The configurator redisplays the configuration setting screen for the `basic_list_value` field with the changed list.

```
        Default:            Current:
            (none)              1) first_value
                                2) second_value_new
```

5. Press **Enter** to set the entries as the value of the list field.

```
cray_example.settings.basic_example.data.basic_list_value
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ <cr>
```

   IMPORTANT: When finished editing list entries, always press **Enter** (<cr>) to set the entries as the value of the list field. If the save and exit command (**Q**) is issued before the entries are set, the configurator will not save the entries and will not write out the new data.

5.4 Change a Multival Setting Field during a Configurator Session

Prerequisites

This procedure assumes that the config set to be updated already exists and the name and location (service) of the setting to be changed is known. If the name or location is not known, see *Locate a Configuration Parameter in a Config Set* on page 64.

About this task

This procedure describes how to change a multival setting using the example of changing setting `multival_example` in service `cray_example` in config set `p0.staging`. A multival setting defines what each subsetting (entry) comprises. In this example, each entry is a `hostname`, and each `hostname` has three fields that need to be configured:

the key that identifies the hostname entry

a string value

an integer value

Procedure

1. Invoke the configurator in interactive mode for the service that contains the setting.

```
smw# cfgset update --mode interactive --service cray_example p0.staging
```

The configurator displays the Service Configuration Menu (service menu). The setting to be changed is #4 in the list of settings.

```
Service Configuration Menu (Config Set: p0.staging, type: cle)

  cray_example        [ status: enabled ]  [ validation: valid ]

--------------------------------------------------------------------------------
  Selected     #      Settings                    Value/Status (level=basic)
--------------------------------------------------------------------------------
                      basic_example
               1)       basic_string_value       def
               2)       basic_protected_value    *** <hidden> ***
               3)       basic_list_value         first_value, second_value
               4)     multival_example
                         hostname: foo           [ OK ]
                         hostname: bar           [ OK ]
--------------------------------------------------------------------------------
**** Select Options ****
  a: all                      n: none                    c: configured
  u: unconfigured             #: toggle #

**** Actions on Selected (1 settings) ****
  C: configure                @: show guidance

**** Other Actions ****
  ?: help                     l: switch level            E: toggle enable
  I: toggle inherit           ^^: go to service list     r: refresh
  $: view changelog           Q: save & exit             x: exit without save

Cray Example Service Menu [default: save & exit - Q] $
```

2. Select the setting/field to be changed.

```
Cray Example Service Menu [default: save & exit - Q] $ 4
```

The configurator redraws the service menu with an asterisk next to the selected item and a different prompt.

3. Configure the selected setting.

Enter **C** or press **Enter** (<cr>) to accept the default action of `configure - C`.

```
Cray Example Service Menu [default: configure - C] $ C
```

The configurator displays the configuration setting screen for the multival_example setting with the fields for each of the two entries ("foo" and "bar").

```
************ cray_example.settings.multival_example ************

      multival_example
        Multiple hostnames can be configured in this service. For each
        hostname, a few simple values are available for further
        configuration.

    Configured Values:
        1) 'foo'
            a) basic_string_value: foovalue
            b) basic_protected_value: *** <hidden> ***
            c) basic_list_value:
                  foovalue1
```

```
                    foovalue2

        2) 'bar'
            a) basic_string_value: barvalue
            b) basic_protected_value: *** <hidden> ***
            c) basic_list_value:
                    barvalue1
                    barvalue2

    Inputs: menu commands (? for help)

|--- Information
|  *    Multiple 'multival_example' entries can be added using this menu
|---

cray_example.settings.multival_example
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $
```

For this menu, the configurator offers the following options to modify the multival entries:

- **+** Add a new entry to the multival setting.
- **<#>*** Modify the # entry, where # is the alphanumeric identifier of a multival entry (subsetting) and field (e.g., **2c*** would select the `basic_list_value` field of the 'bar' entry for modification).
- **<#>-** Delete the entire # multival entry from the list (e.g., **2-** would delete the entire 'bar' entry, and **d** delete all entries (both 'foo' and 'bar').

4. Select the multival entry to be changed.

```
cray_example.settings.multival_example
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ 2a*
```

The configurator redraws the service menu with an asterisk next to the selected item and a different prompt.

5. Configure the selected entry.

Enter **C** or press **Enter** (<cr>) to accept the default action of `configure - C`.

```
Cray Example Service Menu [default: configure - C] $ C
```

The configurator displays the configuration setting screen for the `basic_string_value` field of the 'bar' entry.

```
** cray_example.settings.multival_example.data.bar.basic_string_value **

      basic_string_value -- Basic String Value
        This field is part of the basic_example setting in the cray_example
        service. It is a string value that only allows lowercase letters. The
        configurator will validate the value for this field that it is a string
        and matches the letters only validation criteria. Its default value is
        'abc'.

        Default:          Current:
            none              barvalue

    Value: string, blank values not allowed, regex=^[a-z]*$
           level=basic, state=set

    Inputs: <string>  -- OR --  menu commands (? for help)
```

```
cray_example.settings.multival_example.data.bar.basic_string_value
[<cr>=keep 'barvalue', <new value>, ?=help, @=less] $
```

The user can now make the desired changes to this field.

6. Enter the new value, then press **Enter** to set the value of that field.

```
cray_example.settings.multival_example.data.bar.basic_string_value
[<cr>=keep 'barvalue', <new value>, ?=help, @=less] $ new_barvalue
```

5.5 Change the Service Enabled/Disabled Status during a Configurator Session

Prerequisites

This procedure assumes that the config set to be updated already exists.

About this task

This procedure describes how to configure a service as enabled or disabled using config set `p0.staging` as an example.

Procedure

1. Invoke the configurator in interactive mode.

```
smw# cfgset update --mode interactive p0.staging
```

The configurator displays the Service List Menu.

```
Service Configuration List Menu (Config Set: p0.staging, type: cle)

--------------------------------------------------------------------------------
  Selected     #      Service           Status (level=basic, state=unset)
--------------------------------------------------------------------------------
               1)     cray_example      [ OK ]
               2)     cray_example2     disabled, 7/7 unconfigured settings
--------------------------------------------------------------------------------

**** Select Options ****
  a: all                       n: none                     c: configured
  u: unconfigured              e: enabled                  d: disabled
  i: inheriting                #: toggle #

**** Actions on Selected (0 services) ****
  C: configure                 E: toggle enable            I: toggle inherit
  @: show guidance

**** Other Actions ****
  ?: help                      l: switch level             s: switch state
  r: refresh                   $: view changelog           Q: save & exit
  x: exit without save
```

```
Service List Menu [default: save & exit - Q] $
```

2. Select the service to enable or disable, then toggle its enabled/disabled status.

   Enter the number of the service to be enabled or disabled, then press **Enter** to select it. At the new prompt, enter **E**, then press **Enter** to toggle its enabled/disabled status.

```
Service List Menu [default: save & exit - Q] $ 2
```

```
Service List Menu [default: configure - C] $ E
```

   Because the `cray_example2` service was disabled, toggling its status has enabled it.

3. Enter **Q** to save changes and exit the configurator.

5.6 Change Service Inheritance during a Configurator Session

Prerequisites

This procedure assumes that the service to be changed is inheritable (i.e., one that has a template of both `cle` and `global` type), and that the service is being configured as part of a `cle` config set.

About this task

This procedure describes how to configure a service to inherit (or not) setting values from its `global` configuration template. It uses the `cray_example` service in config set `p0.staging` as an example.

> **NOTE:** If the service is configured to inherit from its global template, the values of all the other settings/fields in the service will still be saved in the configuration data of that config set. However, the processes that consume the data will ignore those setting/field values. Those values are used for configuration only when the service is configured to *not* inherit.

Procedure

1. Invoke the configurator in interactive mode.

```
smw# cfgset update --mode interactive p0.staging
```

   The configurator displays the Service Configuration List Menu (Service List Menu).

```
Service Configuration List Menu (Config Set: p0.staging, type: cle)

--------------------------------------------------------------------------------
  Selected     #      Service           Status (level=basic, state=unset)
--------------------------------------------------------------------------------
               1)     cray_example      inheriting from global config
               2)     cray_example2     disabled, 7/7 unconfigured settings
--------------------------------------------------------------------------------

**** Select Options ****
  a: all                       n: none                      c: configured
  u: unconfigured              e: enabled                   d: disabled
  i: inheriting                #: toggle #
```

```
**** Actions on Selected (0 services) ****
  C: configure                E: toggle enable            I: toggle inherit
  @: show guidance

**** Other Actions ****
  ?: help                     l: switch level             s: switch state
  r: refresh                  $: view changelog           Q: save & exit
  x: exit without save

Service List Menu [default: save & exit - Q] $
```

2. Select the service to change, then toggle its inherit status.

   Enter the number of the service to be changed, then press **Enter** to select it. At the new prompt, enter **I**, then press **Enter** to toggle its inherit status.

```
Service List Menu [default: save & exit - Q] $ 1
```

```
Service List Menu [default: configure - C] $ I
```

3. Enter **Q** to save changes and exit the configurator.

5.7 Revert a Field to its Default Value during a Configurator Session

Prerequisites

This procedure assumes that the config set to be updated already exists and the setting to be changed has a default value.

About this task

This procedure describes how to revert a configuration field to its default value using the example of reverting field `basic_string_value` in service `cray_example` in config set `p0.staging`.

Procedure

1. Invoke the configurator in interactive mode.

```
smw# cfgset update --mode interactive --service cray_example p0.staging
```

   The configurator displays the service menu for the specified service.

2. Select the setting/field to be changed and open the configuration setting screen.

   Enter the number of the setting/field to be changed, then press **Enter** to select it. At the new prompt, enter **C**, then press **Enter** to display the configuration setting screen.

```
Service List Menu [default: save & exit - Q] $ 1
```

```
Service List Menu [default: configure - C] $ C
```

```
****** cray_example.settings.basic_example.data.basic_string_value ******
```

```
      basic_string_value -- Basic String Value
        This field is part of the basic_example setting in the cray_example
        service. It is a string value that only allows lowercase letters. The
        configurator will validate the value for this field that it is a string
        and matches the letters only validation criteria. Its default value is
        'abc'.

        Default:          Current:
            abc               def

    Value: string, blank values not allowed, regex=^[a-z]*$
           level=basic, state=unset

    Inputs: <string>  -- OR --  menu commands (? for help)

cray_example.settings.basic_example.data.basic_string_value
[<cr>=keep 'def', <new value>, ?=help, @=less] $
```

Note that although there is a current value shown and the prompt uses `keep`, the state is shown as unset. This can occur if the template for this service provided prepopulated data for this setting that was different than the default.

3. Revert the field to its default value.

   Enter **#** at the prompt or enter the default value, which is displayed in the middle of the configuration setting screen.

   ```
   cray_example.settings.basic_example.data.basic_string_value
   [<cr>=keep 'def', <new value>, ?=help, @=less] $ #
   ```

4. Enter **Q** to save changes and exit the configurator.

6 Common Tasks When Using Configuration Worksheets for Bulk Import

Configuration worksheets are a bulk upload alternative to the configurator user interface for gathering configuration data and placing it into the config set. Worksheets are recommended when large amounts of data are required or when preparing the data for a config set beforehand is desired. Configuration worksheets contain documentation about their format, contents, and how to edit them. The information and examples provided here supplement that documentation.

- Worksheets are valid YAML files. Invalid YAML syntax in any worksheet file that is input to the config set will cause the config set creation or update to fail.
- Worksheets contain the same guidance and validation information (as comments) that is displayed in the configurator user interface.
- Default values for settings are presented as commented values in the worksheet unless the configuration template has marked them as pre-populated, configured data. To set a value in the config set, uncomment the value in the worksheet. Because the configurator not only sets and validates data, but also keeps track of the changes and configuration status of data, values must be uncommented by the user to signify that the value should be set.
- Information on how to regenerate configuration worksheets and how to create and update config sets using them is provided in the header section of each worksheet and in *Create a Config Set from Configuration Worksheets* on page 19 and *Update a Config Set from Configuration Worksheets* on page 24.

See the following for examples of how to do these common tasks in a configuration worksheet.

6.1 Change the Service Enabled Field in a Configuration Worksheet

Prerequisites

This procedure assumes that the most current configuration worksheet is being edited.

About this task

This procedure describes how to edit a worksheet to configure a service as enabled or disabled using the `cray_example_worksheet.yaml` template as an example.

Procedure

1. Open the worksheet for editing.
2. Move to the **Service Enable/Disable** section.

   If the section is labeled **Service Enable/Inherit** instead, see also *Change the Service Inherit Field in a Configuration Worksheet* on page 77.

   The worksheet begins with general instructions and guidance about editing and using worksheets. Move to the **Service Enable/Disable** section, which provides specific guidance for the `cray_example` service to help the user determine whether to enable the service. The data value for enabling the service is near the end of this section, just below the information that a YAML boolean value is expected for the `cray_example.enabled` field and that its configuration level is basic.

```
 1 #************* Service Enable/Disable **************
 2 # ** INSTRUCTIONS **
 3 #   This section provides the overall description of the service and an
 4 #   enable/disable functionality for the entire service.
 5 #
 6 #   NOTICE: If the service is disabled, the values for all other settings in
 7 #   this worksheet (listed below this section) will be ignored when
 8 #   configuration is applied, for example, at boot time. However, values that
 9 #   are configured will be marked as such in the config set and honored when
10 #   the service is re-enabled.
11 #
12 # Guidance for cray_example (Cray Example Service):
13 #    The Cray Example service is used as an example service for the purpose of
14 #    demonstrating configurator functionality. The default behavior is to
15 #    disable this service.
16 #
17 #
18 # Enable 'cray_example' Service? (boolean, level=basic)
19 #cray_example.enabled: false
20 #
21 #************* END Service Enable/Disable ************
```

3. Edit the service enabled field.
   - To enable the service, uncomment line 19 and set the value to `true`.

     ```
     18 # Enable 'cray_example' Service? (boolean, level=basic)
     19 cray_example.enabled: true
     ```

   - To disable the service, uncomment line 19 and leave the value as `false`.
   - To leave the value for the service enable field unconfigured so that the configurator will prompt users in subsequent UI sessions, leave the field commented.

6.2 Change the Service Inherit Field in a Configuration Worksheet

Prerequisites

This procedure assumes that the most current configuration worksheet is being edited and that the service to be changed is inheritable (i.e., one that has a template of both `cle` and `global` type).

About this task

This procedure describes how to edit a worksheet to configure a service to inherit (or not) setting values from its `global` configuration template. It uses the `cray_example_worksheet.yaml` worksheet as an example.

> **NOTE:** If the service is configured to inherit (inherit field set to `true`), the values of all the other settings/fields in the worksheet will still be read into the configurator and saved in the configuration data of that config set. However, the processes that consume the data will ignore those setting/field values. Those values are used for configuration only when the service is configured to *not* inherit (inherit field set to `false`).

Procedure

1. Open the worksheet for editing.
2. Move to the **Service Enable/Inherit** section.

```
1 #************** Service Enable/Inherit **************
 2 # ** INSTRUCTIONS **
 3 #   This section provides the overall description of the service and an
 4 #   enable/disable functionality for the entire service.
 5 #
 6 #   NOTICE: If the service is disabled, the values for all other settings
 7 #   in this worksheet (listed below this section) will be ignored when
 8 #   configuration is applied, for example, at boot time. However, values
 9 #   that are configured will be marked as such in the config set and honored
10 #   when the service is re-enabled.
11 #
12 # ** INHERITANCE **
13 #  The values in this service are inheritable, meaning they can be
14 #  overridden by those contained within a 'global' config set version.
15 #  Would you like to inherit values? This value should remain false if this
16 #  worksheet is being configured for a global config set. If the service is
17 #  inherited, the values for all other settings in this worksheet (listed
18 #  below this section) will be ignored at boot time. Values that are
19 #  configured will be marked as such in the config set and honored when the
20 #  service is enabled and no longer inherited.
21 #
22 #
23 # Inherit Values for service cray_example from a Global Config Set? (boolean)
24 cray_example.inherit: false
25 #
26 # Guidance for cray_example (Cray Example Service):
27 #    The Cray Example service is used as an example service for the purpose
28 #    of demonstrating configurator functionality. The default behavior is
29 #    to disable this service.
30 #
31 #
32 # Enable 'cray_example' Service? (boolean, level=basic)
33 cray_example.enabled: true
34 #
35 #************ END Service Enable/Inherit **************
```

3. Edit the service inherit field.
   - To inherit values for the service from a `global` config set version of the service, set the value in line 24 to `true`.

- To disable inheritance, leave the value in line 24 as `false`.
- To leave the value for the service inherit field unconfigured so that the configurator will prompt users in subsequent UI sessions, comment out line 24.

6.3 Change a Basic Setting Field in a Configuration Worksheet

Prerequisites

This procedure assumes that the most current configuration worksheet is being edited.

About this task

This procedure describes how to change a basic setting field (i.e., a field of type string, integer, float, or boolean) in a worksheet using the example of changing the `basic_string_value` field in the `cray_example_worksheet.yaml` worksheet.

Procedure

1. Open the worksheet for editing.
2. Move to the **START Service Setting: basic_example** section.

   The `basic_example` setting has two fields: `basic_string_value` (starts at line 11) and `basic_integer_value` (starts at line 24). The `basic_string_value` field on line 22 is unconfigured (because it is commented out), is currently set to its default value, and has configuration level basic. The `basic_integer_value` field on line 33 is configured (because it is uncommented), is set to a value other than its default, and has configuration level advanced.

```
 1 #*********** START Service Setting: basic_example **********
 2 # ** INSTRUCTIONS **
 3 #   This section describes configuration values for the 'basic_example'
 4 #   setting. Uncomment and change (if desired) a value to configure it.
 5 #
 6 #   If a field is uncommented in the original worksheet, this value has been
 7 #   preconfigured by the service since it is expected to be present to
 8 #   properly configure the system. It is recommended that the uncommented
 9 #   data is kept as specified in this file.
10 #
11 #----------- basic_example : basic_string_value -----------
12 # Basic String Value (type=string, level=basic)
13 # Guidance:
14 #    This field is part of the basic_example setting in the cray_example
15 #    service. It is a string value that only allows lowercase letters. The
16 #    configurator will validate the value for this field that it is a string
17 #    and matches the letters-only validation criteria. Its default value
18 #    is 'abc'.
19 #
20 # Validation: regex=^[a-z]*$
21 #
22 #cray_example.settings.basic_example.data.basic_string_value: abc
23 #
24 #----------- basic_example : basic_integer_value ----------
25 # Basic Integer Value (type=integer, level=advanced)
```

```
26 # Guidance:
27 #    This field is part of the basic_example setting in the cray_example
28 #    service. It is a integer value that only allows numbers. The
29 #    configurator will validate the value for this field that it is an
30 #    integer. Its default value is 123.
31 #
32 #
33 cray_example.settings.basic_example.data.basic_integer_value: 456
34 #
35 #*********** END Service Setting: basic_example ***********
```

3. Set the `basic_string_value` field.

   To set the `basic_string_value` field, uncomment line 22. Change the value, if desired.

6.4 Change a Multival Setting Field in a Configuration Worksheet

Prerequisites

This procedure assumes that the most current configuration worksheet is being edited.

About this task

This procedure describes how to modify a multival setting in a worksheet using the example of changing the `multival_example` setting in the `cray_example_worksheet.yaml` worksheet. A multival setting defines what each subsetting (entry) comprises. In this example, each entry is a `hostname`, and each `hostname` has three fields that need to be configured:

the key that identifies the hostname entry
a string value
an integer value

The configurator schema supports the case where multival settings can be embedded within other multival settings. In this case, the configuration worksheet will provide guidance for configuring these settings as it does for non-embedded multival settings.

CAUTION: When editing worksheets for config sets that have already been created, any multival data previously configured will already be present in the worksheet. If any of this data is removed from the worksheet, then when the worksheet is used to update the config set, the data removed from the worksheet will also be removed from the config set.

Procedure

1. Open the worksheet for editing.

2. Move to the **START Service Setting: multival_example** section.

   The configuration worksheet provides guidance for the overall `multival_example` setting, an example of how to specify a `multival_example` entry (lines 21–24), and then provides guidance, validation criteria, and specification of each field in an entry. If any entries for the `multival_example` setting had been configured, they would be located at the end of the setting section (line 72).

```
#******** START Service Setting: multival_example *********
# ** INSTRUCTIONS **
#   This section describes configuration values for the 'multival_example'
#   setting. Multiple values of 'multival_example' can be configured in
#   this worksheet by giving each entry a unique key identifier called a
#   'multival key'. Each entry must have a unique multival key field to
#   differentiate its values from others that may be configured. The
#   multival key field for the 'multival_example' setting is 'hostname'.
#   Some settings may also contain multiple levels of embedded settings
#   within their definition. The example 'multival_example' value shown
#   below will provide the proper structure for these embedded fields if
#   they exist for the current setting.
#
#   Specify values for 'multival_example' setting at the end of this section.
#
# ** GUIDANCE FOR 'multival_example' SETTING:
#    Multiple hostnames can be configured in this service. For each
#    hostname, a string value and an integer value are available for
#    further configuration.
#
# ** EXAMPLE 'multival_example' VALUE  (with current defaults) **
#   cray_example.settings.multival_example.data.hostname.sample_key_a: null
                                                        <-- setting a multival key
#   cray_example.settings.multival_example.data.sample_key_a.basic_string_value: abc
#   cray_example.settings.multival_example.data.sample_key_a.basic_integer_value: 123
#
#
# ** 'multival_example' FIELD SPECIFICATION -- MULTIVAL KEY FIELD **
#-------------- multival_example : hostname -----------
# Hostname (type=string, level=basic)
# Guidance:
#    Provide a hostname for this example multival key field. The values for
#    this field will be keys in the data structure.
#
#
# Default Value:
#  hostname: hostname_foo
#
#
# ** 'multival_example' FIELD SPECIFICATION -- OTHER FIELDS **
#--------- multival_example : basic_string_value -----------
# Basic String Value (type=string, level=basic)
# Guidance:
#    This field is part of the basic_example setting in the cray_example
#    service. It is a string value that only allows lowercase letters. The
#    configurator will validate the value for this field that it is a string
#    and matches the letters only validation criteria. Its default value
#    is 'abc'.
#
# Validation: regex=^[a-z]*$
#
# Default Value:
#  basic_string_value: abc
#
#----------- multival_example : basic_integer_value ---------
# Basic Integer Value (type=integer, level=advanced)
# Guidance:
#    This field is part of the basic_example setting in the cray_example
#    service. It is a integer value that only allows numbers. The
#    configurator will validate the value for this field that it is an
#    integer. Its default value is 123.
#
#
# Default Value:
#  basic_integer_value: 123
#
#
# ** END 'multival_example' FIELD SPECIFICATION
#
# ** 'multival_example' DATA **
#
# NOTE: Place additional 'multival_example' setting entries here, if desired.

#********* END Service Setting: multival_example *********
```

3. Add an entry for the `multival_example` setting.

a. Copy lines 22–24, paste them at line 72, and uncomment them.

```
72 cray_example.settings.multival_example.data.hostname.sample_key_a: null
                                                    <-- setting a multival key
73 cray_example.settings.multival_example.data.sample_key_a.basic_string_value: abc
74 cray_example.settings.multival_example.data.sample_key_a.basic_integer_value: 123
```

b. Replace `sample_key_a` in lines 72–74 with the key for the new entry, and remove the `<-- setting a multival key` text at the end of line 72.

Note that the 'null' value in line 72 is required. Do not remove or change it. This is true for all multival entry keys.

```
72 cray_example.settings.multival_example.data.hostname.foo: null
73 cray_example.settings.multival_example.data.foo.basic_string_value: abc
74 cray_example.settings.multival_example.data.foo.basic_integer_value: 123
```

c. Replace the values of `basic_string_value` in line 73 and `basic_integer_value` in line 74 with the desired values.

```
73 cray_example.settings.multival_example.data.foo.basic_string_value: foo_value
74 cray_example.settings.multival_example.data.foo.basic_integer_value: 456
```

4. Repeat step 3 to add another entry for the `multival_example` setting with the following values: hostname key = 'bar,' string = 'bar_value,' and integer = '789.'

```
67 # ** END 'multival_example' FIELD SPECIFICATION
68 #
69 # ** 'multival_example' DATA **
70 #
71 # NOTE: Place additional 'multival_example' setting entries here, if desired.
72 cray_example.settings.multival_example.data.hostname.foo: null
73 cray_example.settings.multival_example.data.foo.basic_string_value: foo_value
74 cray_example.settings.multival_example.data.foo.basic_integer_value: 456
75
76 cray_example.settings.multival_example.data.hostname.bar: null
77 cray_example.settings.multival_example.data.foo.basic_string_value: bar_value
78 cray_example.settings.multival_example.data.foo.basic_integer_value: 789
79 #********** END Service Setting: multival_example *********
```

7 cfgset Troubleshooting Tips

Unable to Update a Service in a Config Set

The following command to update *SERVICE* in config set `p0` can result in a variety of outcomes, depending on the level and state of the settings in that service.

```
smw# cfgset update --service SERVICE p0
```

Note that for a service with configuration template file `cray_example_config.yaml`, use only the `cray_example` portion on the command-line when specifying a single service.

- **Outcome 1: No configuration settings presented.**

  ```
  INFO - Running pre-configuration scripts
  ...
  INFO - Merging configuration templates and validating schema.
  INFO - Configuration worksheets will be saved to /var/opt/cray/imps/config/
  sets/p0/worksheets
  INFO - Changelog will be written to
         - /var/opt/cray/imps/config/sets/p0/changelog/
  changelog_2015-12-02T16:39:25.yaml
  INFO - Running post-configuration scripts
  ...
  INFO - ConfigSet 'p0' has been updated.
  ```

  The command does not specify mode, level, or state, so defaults are used: `auto` mode, level `basic`, and state `unset`. Therefore, the configurator looks only for required and basic settings that are unset. If it finds none, no interaction with the user is necessary, so it proceeds directly to saving worksheets and logs, and then `cfgset` runs post-configuration activities and exits automatically. **If the intention was to confirm that all required and basic settings have been set, then this is the desired outcome.** However, if the intention was to view all settings and perhaps change a few, use this command instead:

  ```
  smw# cfgset update --service SERVICE --level advanced --mode interactive p0
  ```

- **Outcome 2: Some configuration settings presented, but not the ones that need to be changed.**

  The settings that need to be set/changed are not presented because either they are already set or they are level `advanced`. Try this:

  1. Enter **^** at the configurator prompt to switch to `interactive` mode. Now settings of all states are displayed in the service menu and can be selected and set/changed. If the desired settings are still not found in the service menu, continue to the next step.
  2. Enter **l** (lowercase L) at the configurator prompt to switch to the next level (cycles through all three levels) until `level=advanced` displays in the service menu header. Now settings of all levels and states are displayed in the service menu and can be selected and set/changed.

- **Outcome 3: Some new and unfamiliar configuration settings presented.**

If the service package that contains the service being updated has been reinstalled, the associated service configuration template may have new or revised settings and values. The configurator will find that template in `/opt/cray/imps_config/`*`SERVICE_PACKAGE`*`/default/configurator/template` and merge its contents with configuration data already in the config set. When the configurator presents those new settings to the user, they may appear unfamiliar. If settings other than the ones presented need to be set/changed, see Outcome 2.

Validation Rule Failure

When `cfgset validate` encounters a rule failure, a non-zero value is returned and the rule failure is printed:

```
smw# cfgset validate p0
...
Validating ConfigSet 'p0'

Lookup/Reference Errors (1):
    Template: /var/opt/cray/imps/config/sets/p0.alison/config/cray_dvs_config.yaml
        Error: The configured value 'dvs_servers' is not located in the reference
field 'cray_node_groups.settings.groups'
        Location: cray_dvs.settings.client_mount.data.test-ro.server_groups
```

Rule failure can be remedied by adjusting config set data to conform with the failed rule. Alternatively, the rule can be temporarily bypassed using either the `--no-rules` or `--exclude-rule` option. See *Validate a Config Set and List Validation Rules* on page 38 for more details on bypassing validation rules.