CSR

**MASSACHUSETTS INSTITUTE OF TECHNOLOGY**
**CENTER FOR SPACE RESEARCH**
**CAMBRIDGE, MASSACHUSETTS 02139**

**REVISION LOG**

**TITLE: Software Detailed Design**
**System Configuration Classes**

**DOC. NO.**
**36-53238 Rev. A**

| Revision | Date (mm/dd/yy) | ECO No. | Page(s) Affected | Reason | Approval |
|---|---|---|---|---|---|
| A | 4/10/96 | 36-578 | all | Initial version. Incorporated comments from initial review. | 4/22/96 |

# 30.0 System Configuration Classes (36-53238 A)

## 30.1 Purpose

The purpose of the System Configuration classes is to manage the suite of software-controllable hardware settings within the instrument.

## 30.2 Uses

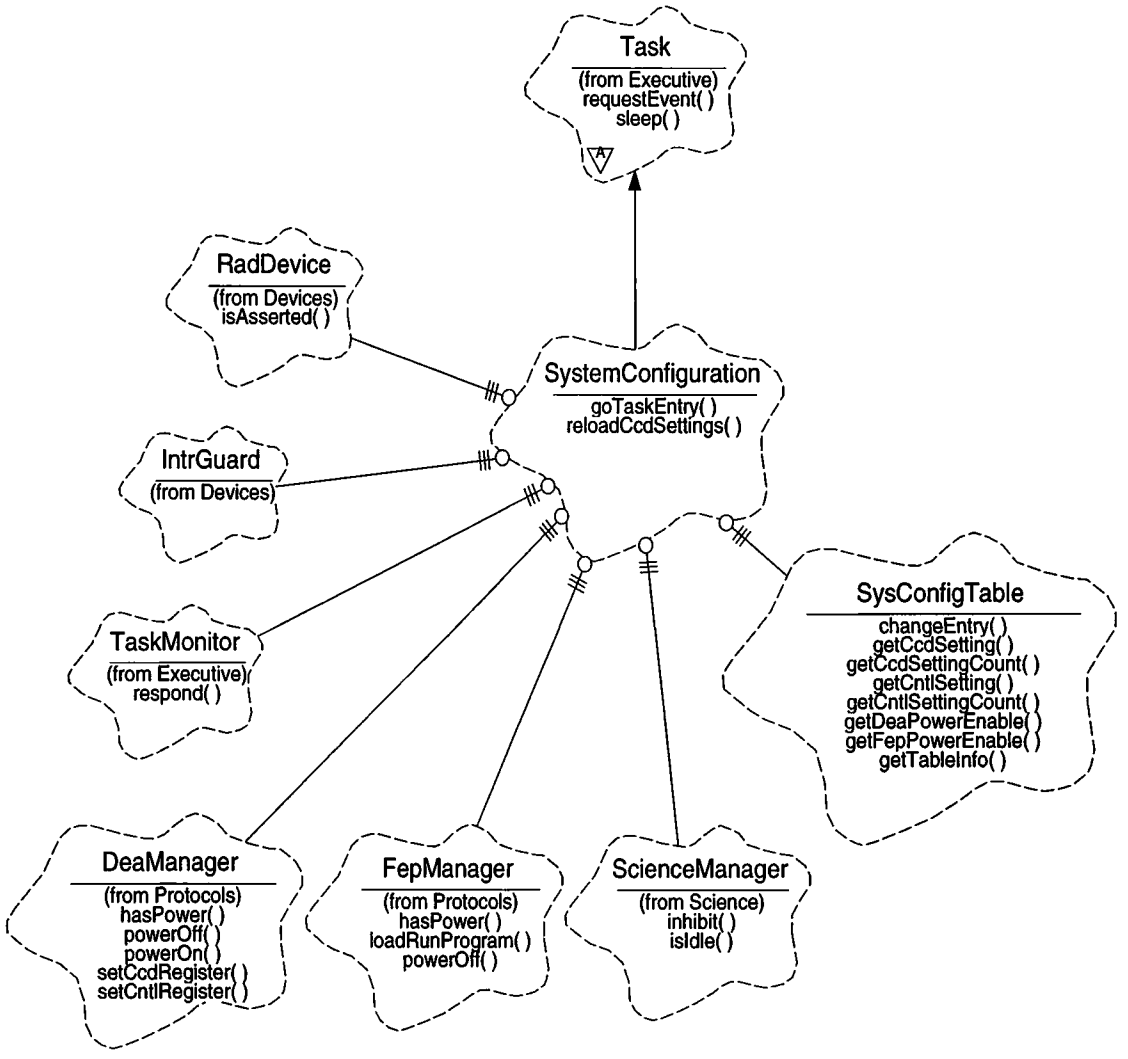
The following lists the use of the System Configuration Classes:

Use 1:: Update settings within the system's table
Use 2:: Update DEA and FEP power selections
Use 3:: Load updated settings into the DEA interface controller
Use 4:: Power off the DEA if the radiation monitor is asserted
Use 5:: Re-enable DEA power once the radiation monitor de-asserts
Use 6:: Re-load settings into a DEA CCD controller

## 30.3 Organization

Figure 131 illustrates the class relationships used by the System Configuration Classes. These class include the **`SysConfigTable`** class, which maintains the current list of settings, and the **`SystemConfiguration`** class, which is a task which reloads settings into the hardware and monitors the radiation flag.

**FIGURE 131. System Configuration Class Relationships**




**SysConfigTable**- This class represents the collection of system configuration parameters within the instrument. It provides a function which modifies an entry in the table (`changeEntry`), and provides a function which returns the address and length of the table for purposes of dumping the table to telemetry (`getTableInfo`). This class provides services used by the **SystemConfiguration** class to get the desired power settings for each of the DEA and FEP boards (`getDeaPowerEnable`, `getFepPowerEnable`), to fetch a desired DEA CCD Controller or Interface Controller register setting from the table (`getCcdSetting`, `getCntlSetting`, respectively), and to determine the total number of settings for the CCD Controller or Interface Controller (`getCcdSettingCount`., `getCntlSettingCount`, respectively).

**SystemConfiguration**- This class is a subclass of ***Executive*::Task**, and is responsible for maintaining the hardware configuration settings and reacting to the radia-

tion monitor. This class provides a main task entry function (`goTaskEntry`) and a public function used to reload all settings into a CCD Controller (`reloadCcdSettings`).

**`Task`**- This class is supplied by the ***`Executive`*** class category. It represents and controls an active running task. The **`SystemConfiguration`** class inherits from this class, and uses the class's functions to relinquish control for a period of time (`sleep`), and to detect queries from the TaskMonitor (`requestEvent`).

**`RadDevice`**- This class is supplied by the ***`Devices`*** class category. It is responsible for providing access to the radiation monitor flag (`isAsserted`).

**`IntrGuard`**- This class is supplied by the ***`Devices`*** class category and is responsible for disabling and re-enabling interrupts.

**`TaskMonitor`**- This class is supplied by the ***`Executive`*** class category, and is responsible for periodically polling each task in the instrument. When polled, the **`SystemConfiguration`** task responds using this classes member function (`respond`).

**`DeaManager`**- This class is supplied by the ***`Protocols`*** class category, and is responsible for arbitrating access to the DEA hardware interface. The **`SystemConfiguration`** class uses this class to disable power to the DEA boards (`powerOff`), to query the current power state of the boards (`hasPower`), to enable power to the boards (`powerOn`), and to write to the CCD Controller and Interface Controller boards registers (`setCcdRegister`, `setCntlRegister`, respectively).

**`FepManager`**- This class is supplied by the ***`Protocols`*** class category, and is responsible for managing access to the Front End Processors. The **`SystemConfiguration`** class uses this class to query the current power state of each of the FEPs (`hasPower`), to disable power to a FEP (`powerOff`), and to power on a FEP and load a default program (`loadRunProgram`).

**`ScienceManager`**- This class is supplied by the ***`Science`*** class category, and is responsible for managing overall science processing. The **`SystemConfiguration`** class uses this class to detect when a science run is complete (`isIdle`), and to abort a run in-progress and inhibit further runs (`inhibit`).

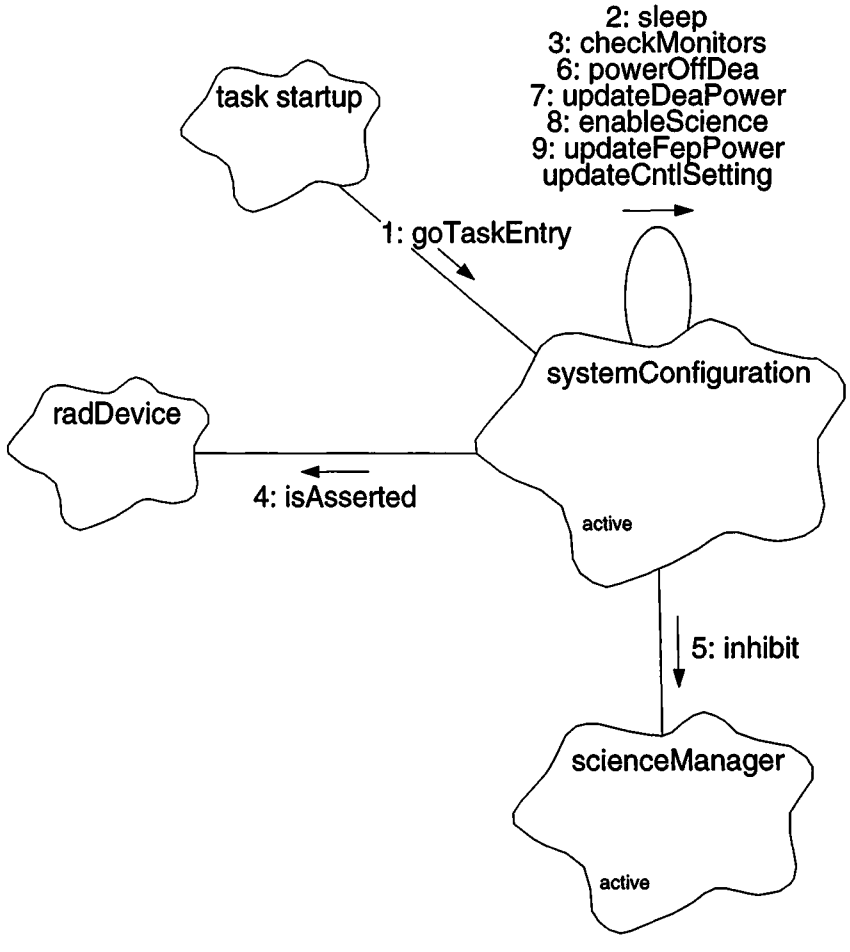
## 30.4 Scenarios

### 30.4.1 Overall SystemConfiguration task operation

The **`SystemConfiguration`** task consists of a main polling loop, contained within `goTaskEntry()`, which uses the **`RadDevice`** class to determine when the radiation monitor is asserted.

Figure 132 illustrates the main operations performed on each iteration of this loop, and the interactions with the Radiation Monitor interrupt.

**FIGURE 132. SystemConfiguration task polling loop**




1. During system startup, the executive starts the *`systemConfiguration`* task using *`systemConfiguration`*`.goTaskEntry()`. The task function `goTaskEntry()` then enters its infinite polling loop.
2. At the top of its loop, `goTaskEntry()` calls `sleep()` to allow other tasks to run.
3. It then checks for, and responds to queries from the *`taskMonitor`*, and checks the current state of the radiation monitor using `checkMonitors()`.

4. It then checks if the radiation monitor has been asserted, using radDevice.isAsserted(). Assume for this example, that the monitor has been asserted.
5. Once the task detects the assertion of the monitor, it tells science to end its current run and delay the start of any subsequent runs until the radiation condition subsides, by passing *BoolTrue* to *scienceManager*.inhibit().
6. The task then shuts off power to all of the DEA's CCD-controller boards, using powerOffDea().
7. If the radiation monitor is not asserted, the task then updates any changes to the DEA power settings using updateDeaPower().
8. If the monitor had previously been asserted, the task then enables science runs using enableScience(), which then passes *BoolFalse* to *scienceManager*.inhibit() to perform the enable (and possibly re-start a previously shutdown science run).
9. On each iteration, the task updates the FEP power settings using updateFepPower(), and updates the DEA Interface Controller settings using updateCntlSettings().

### 30.4.2 Use 1: Update settings within the system's table

To modify an entry in the system configuration table, the client passes the entry index and value to *sysConfigTable*.changeEntry(), which then modifies the indexed value and sets the corresponding changed flag to *BoolTrue*. Later, the *systemConfiguration* task will pick up the modification, adjust the corresponding DEA hardware setting or power selection as described in the subsequent scenarios and sets the changed flag to *BoolFalse*.

The final layout of the table is TBD. Table 25 illustrates the current layout:

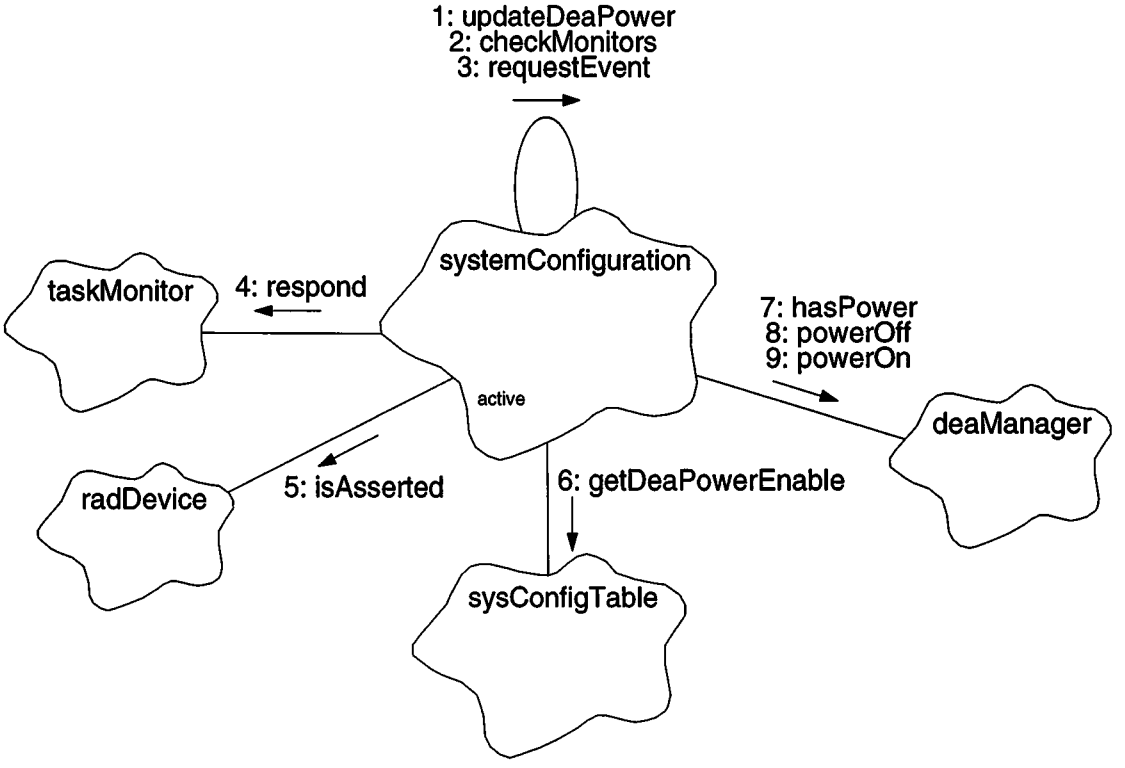
**TABLE 25. System Configuration Table Layout**

| Index | Name | Quantity | Description |
|---|---|---|---|
| 0 | DEA Power | 1 | This is a bit-field, indicating which CCD-controllers should be powered. Each bit is indexed by CCD Id (i.e. bit 0 corresponds to CCD_I0). If a bit is '1', then the CCD controller should be on, otherwise, it should be off. Unused bits in the field should be set to zero. |
| 1 | FEP Power | 1 | This is a bit-field, indicating which FEPs should be powered. Each bit is indexed by FEP Id (i.e. bit 0 corresponds to FEP_0). If a bit is '1', then the FEP should be on, otherwise, it should be off. Unused bits in the field should be set to zero. |
| 2..65 | DEA Interface Controller Settings | 64 | This is an array of interface board settings. The assignments are TBD. |
| 66+ (64 * *ccdId*) + *settingId* | DEA CCD Controller Settings | 10 * 64 | This is a set of arrays of CCD Controller settings. The index formula indicates the offset to a particular setting, indicated by *settingId* for a particular CCD Controller, indicated by *ccdId*. The register assignments are TBD. |

### 30.4.3 Use 2: Update DEA and FEP power selections

Figure 133 illustrates the case where the **SystemConfiguration** updates the DEA power settings.

**FIGURE 133. Update DEA Power Settings**




1. The main polling loop of the *systemConfiguration* task determines that the radiation monitor is off. It takes the opportunity to update any DEA power settings which have changed by calling updateDeaPower().
2. updateDeaPower() consists of two loops. The first loop powers off any DEA boards which have been disabled. The second loop then powers on any boards which have been enabled (NOTE: The settings for a given CCD controller are re-loaded at the onset of the next science run). At the top of each loop iteration, updateDeaPower() calls checkMonitors() to ensure that the *taskMonitor* and radiation monitors are responded to in a timely fashion.
3. checkMonitors() first checks to see if a query from the *taskMonitor* is outstanding, using the inherited function, **Task**::requestEvent().
4. If a query is pending, checkMonitors() replies to the query using *taskMonitor*.respond().
5. checkMonitors() then checks to see if the radiation monitor is active by calling isRadiationOn() (not shown). isRadiationOn() then calls *radDevice*.isAsserted() to test the hardware radiation monitor flag. If the flag is

asserted, then checkMonitors() returns, indicating that the current update operation should be aborted. If the radiation monitor is inactive, then the current operation may proceed. Assume for the remainder of the scenario that the monitor is inactive.

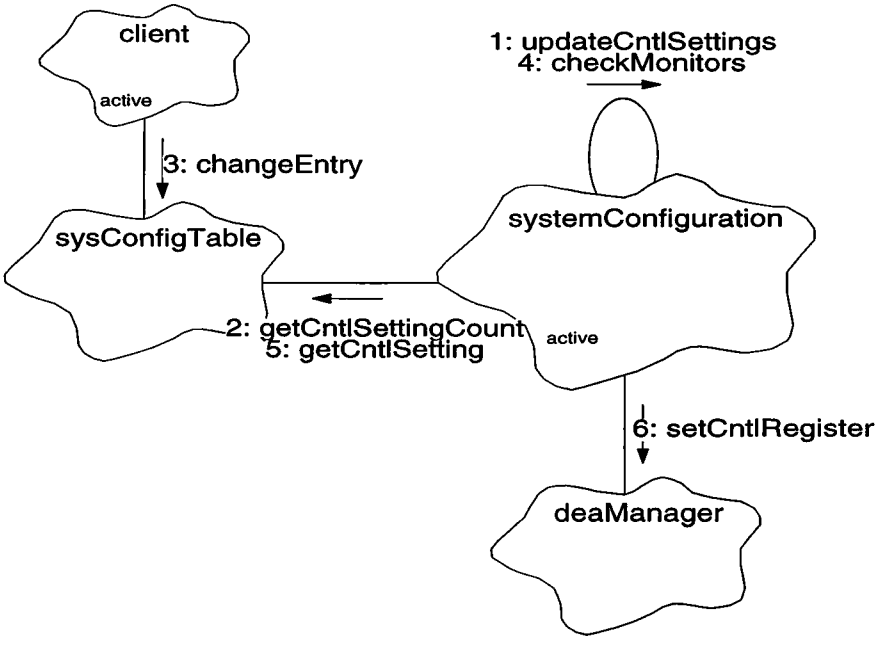
6. updateDeaPower() then obtains the desired power setting for a DEA board using *sysConfigTable*.getDeaPowerEnable().
7. updateDeaPower() then compares this with the current power state of the board, as indicated by *deaManager*.hasPower().
8. If the configuration indicates that the board should be off, but it is on, updateDeaPower() shuts off power to the board using *deaManager*.powerOff().
9. Once all of the boards which should be off have been disabled, updateDeaPower() checks for boards which should be turned on. If a board's power is off when it should be on, updateDeaPower() enables power to the board using *deaManager*.powerOn().

The scenario for powering FEPs is similar to the above scenario except that the desired power settings are obtained using *sysConfigTable*.getFepPowerEnable(), and the queries and power control functions (hasPower, powerOff, powerOn) are issued to the *fepManager*, rather than the *deaManager*. Since changes in FEP power are driven by power-consumption and thermal concerns, power settings to the FEPs are always updated, regardless of the state of the radiation monitor.

### 30.4.4 Use 3: Load updated settings into the DEA interface controller

Figure 134 illustrates the steps used by the **SystemConfiguration** class to update modified register settings in the DEA interface controller board.

**FIGURE 134. Load updated DEA settings**

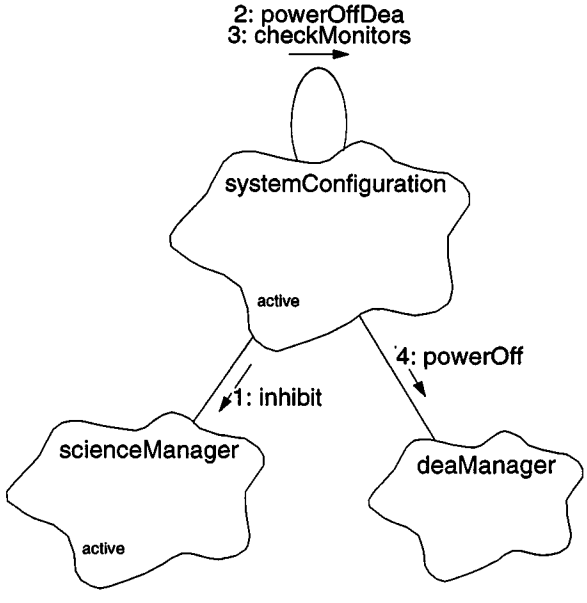



1. The *systemConfiguration*'s polling loop calls updateCntlSettings() to update any DEA interface controller register values which have changed.
2. updateCntlSettings() obtains the total number of settings for the interface board using *sysConfigTable*.getCntlSettingCount(). It then enters its main loop to update the collection of settings.
3. Meanwhile, a client object may make changes to the collection of settings using *sysConfigTable*.changeEntry(). If a change is made to a setting prior to it's being processed by updateCntlSettings(), it will be loaded into the hardware on this invocation of updateCntlSettings(). If, however, a change is made after it has been passed over by updateCntlSettings(), it won't be loaded into the DEA until the next iteration of the *systemConfiguration* task's polling loop.
4. At the start of each iteration of the loop, updateCntlSettings() calls checkMonitors() to respond to queries from the *taskMonitor*.
5. updateCntlSettings() obtains the desired value for a given DEA board's register, whether or not it is used by the hardware, and whether or not the desired setting has changed since the last update, using *sysConfigTable*.getCntlSetting().
6. If the register setting is used, and has changed, updateCntlSettings() loads the value into the DEA board using *deaManager*.setCntlRegister().

### 30.4.5 Use 4: Power off the DEA if the radiation monitor is asserted

Figure 135 illustrates the steps used by the **`SystemConfiguration`** class to shutdown the DEA when the radiation monitor has been asserted.

**FIGURE 135. DEA shutdown due to radiation monitor assertion**




1. When the *`systemConfiguration`* task's polling loop detects that the radiation monitor has gone off, it instructs the *`scienceManager`* to stop its current run, and delay the onset of any subsequent runs, by passing *`BoolTrue`* to *`scienceManager`*.`inhibit`().
2. It then immediately shuts off power to all of the DEA's CCD-controller boards using `powerOffDea`().
3. `powerOffDea`() iterates through each board. At top of each iteration, `powerOffDea`() calls `checkMonitors`() to respond to any queries from the *`taskMonitor`*. Since it already knows that the radiation monitor has been asserted, `powerOffDea`() ignores the return value from `checkMonitors`().
4. As part of its processing loop, `powerOffDea`() then shuts off power to each DEA CCD-controller using *`deaManager`*.`powerOff`().

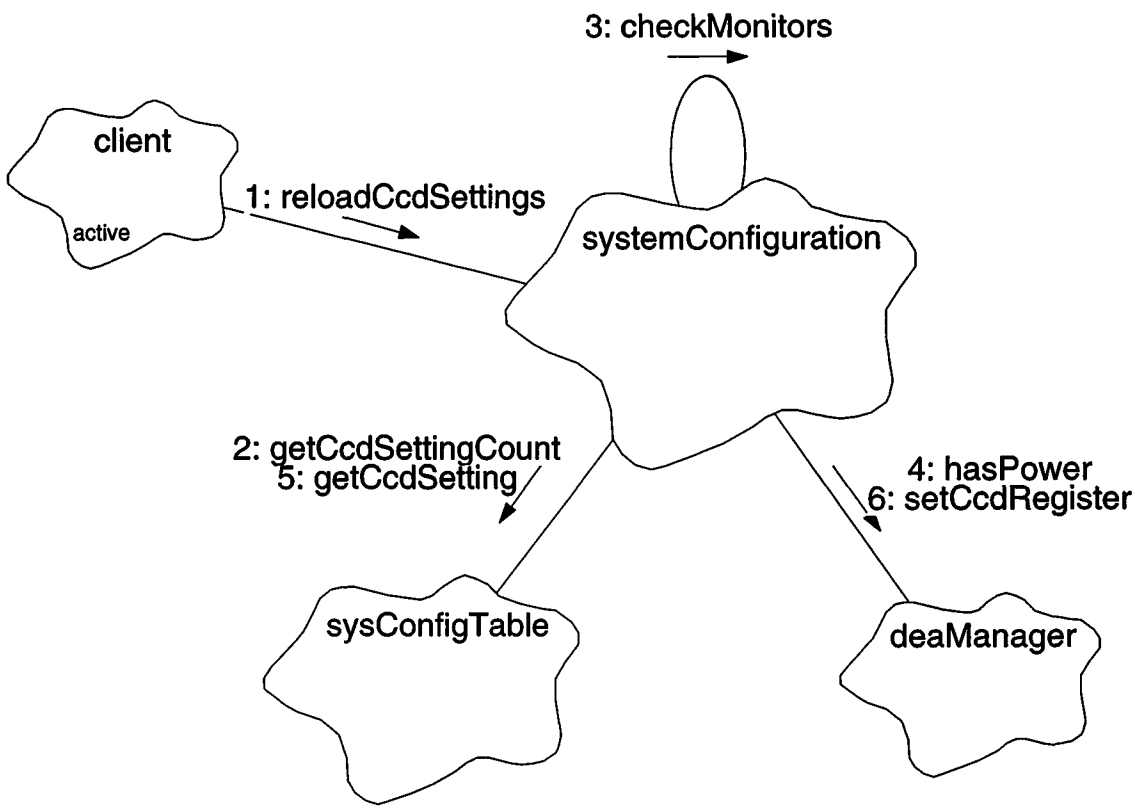
### 30.4.6 Use 5: Re-enable DEA power once the radiation monitor de-asserts

Once the radiation monitor subsides, the *systemConfiguration*'s polling loop calls updateDeaPower() to power on all of the required DEA boards (see Section 30.4.3). Their respective settings are re-loaded at the onset of the next science run. It then enables science by passing *BoolFalse* to *scienceManager*.inhibit().

### 30.4.7 Use 6: Re-load settings into a DEA CCD controller

The settings for a given CCD Controller are re-loaded when the CCD is about to be used for a science run. Its settings are loaded as part of the setup for the run. Figure 136 illustrates the steps used by the SystemConfiguration to load the settings for a particular CCD.

**FIGURE 136. Re-load CCD Controller Settings**




1. The client (*scienceManager*) attempts to re-load the settings for a particular CCD controller using *systemConfiguration*.reloadCcdSettings().
2. reloadCcdSettings() obtains the number of settings used for a CCD controller using *sysConfigTable*.getCcdSettingCount().
3. reloadCcdSettings() then enters loading loop. On each iteration, reloadCcdSettings() calls checkMonitor() to test for and respond to queries from the **TaskMonitor**.

4. `reloadSettings()` checks that the selected CCD controller was commanded to be powered on using *`deaManager`*`.hasPower()`. If the board was not powered on, then the function immediately returns.
5. If the board has power, `reloadCcdSettings()` obtains the next setting using *`sysConfigTable`*`.getCcdSetting()`.
6. It loads the setting into the CCD Controller using *`deaManager`*`.setCcdRegister()`.

## 30.5 Class SysConfigTable

Documentation:

This class acts as a repository for the system configuration settings.

Export Control: Public

Cardinality: 1

Hierarchy:

Superclasses: **none**

Public Interface:

Operations:
```
SysConfigTable()
changeEntry()
getCcdSetting()
getCcdSettingCount()
getCntlSetting()
getCntlSettingCount()
getDeaPowerEnable()
getFepPowerEnable()
getTableInfo()
```

Protected Interface:

Operations: `getSetting()`

Private Interface:

Has-A Relationships:

**Boolean** *inuseCcdMap*[64 (TBD)]: This is an array of in-use flags indicating whether a particular DEA CCD-controller board register is used by the hardware or not. The array is indexed by register index. If an entry is *BoolFalse*, then the corresponding setting is not used by the system. If an entry is *BoolTrue*, then the setting entry is valid.

**Boolean** *inuseCntlMap*[64 (TBD)]: This is an array of in-use flags indicating whether a particular DEA interface board register is used by the hardware or not. The array is indexed by register index. If an entry is *BoolFalse*, then the corresponding setting is not used by the system. If an entry is *BoolTrue*, then the setting entry is valid.

**Boolean** *settingChanged*[2818 (TBD)]: This is an array of flags which indicate which values have been changed since the last read. *BoolFalse* indicates that the value hasn't changed since the last query, and *BoolTrue* indicates that the value has changed. (TBD: use bit-field)

**unsigned short** *setting*[2818 (TBD)]: This is the array of configuration setting values.

Concurrency: Guarded

Persistence: Persistent

### 30.5.1 SysConfigTable()

Public member of: **SysConfigTable**

Documentation:

This function is the constructor for the **SysConfigTable** class.This function initializes the in-use tables, and flags all settings as changed.

Concurrency: Sequential

### 30.5.2 changeEntry()

Public member of: **SysConfigTable**

Return Class: **void**

Arguments:

**unsigned** *item*
**unsigned** *value*

Documentation:

This function modifies the configuration entry, setting the entry indicated by *item* to the passed *value*.

Concurrency: Synchronous

### 30.5.3 getCcdSetting()

Public member of: **SysConfigTable**

Return Class: **Boolean**

Arguments:

**CcdId** *ccdid*
**unsigned** *regid*
**unsigned short&** *value*
**Boolean&** *changed*

Documentation:

This function reads the desired setting for the DEA CCD Controller register. The CCD controller is indicated by *ccdid*, and the register setting is indexed by *regid*. The function returns the desired setting in *value* and resets the modification flag corresponding to the entry. If the value has changed since the last call, the function sets *changed* to *BoolTrue*. If the value hasn't changed since the last time it was fetched, *changed* contains *BoolFalse*. If the register is not used by the DEA hardware, the function returns *BoolFalse*. If it is a valid hardware register, the function returns *BoolTrue*. (NOTE: This scheme is to allow for late changes to the DEA register settings without impacting the software design).

Concurrency: Synchronous

### 30.5.4 getCcdSettingCount()

Public member of: **SysConfigTable**

Return Class: **unsigned**

Documentation:

This function returns the fixed number of CCD controller settings in the table.

Concurrency: Synchronous

### 30.5.5 getCntlSetting()

Public member of: **SysConfigTable**

Return Class: **Boolean**

Arguments:

**unsigned** *regid*
**unsigned short&** *value*
**Boolean&** *changed*

Documentation:

This function reads the desired setting for the DEA Interface Controller register. The register setting is indexed by *regid*. The function returns the desired setting in the value and resets the changed flag corresponding to the entry. If the value has changed since the last call, the function sets *changed* to *BoolTrue*. If the value hasn't changed since the last time it was fetched, *changed* contains *BoolFalse*. If the register is not used by the DEA hardware, the function returns *BoolFalse*. If it is a valid hardware register, the function returns *BoolTrue*. (NOTE: This scheme is to allow for late changes to the DEA register settings without impacting the software design).

Concurrency: Synchronous

### 30.5.6 getCntlSettingCount()

Public member of: **SysConfigTable**

Return Class: **unsigned**

Documentation:

This function returns the fixed number of Interface controller settings in the table.

Concurrency: Synchronous

### 30.5.7 getDeaPowerEnable()

Public member of: **SysConfigTable**

Return Class: **Boolean**

Arguments:

**DeaBoardId** *boardid*

Documentation:

This function returns whether or not the DEA board, indicated by *boardid*, is configured to be powered on. If the board should not be powered on, this function returns *BoolFalse*. If the board should be powered on when the radiation monitor is not active, the function returns *BoolTrue*.

Concurrency: Synchronous

### 30.5.8 getFepPowerEnable()

Public member of: **SysConfigTable**

Return Class: **Boolean**

Arguments:

**FepId** *fepid*

Documentation:

This function indicates whether or not the FEP, indicated by *fepid*, should be powered on. If the function returns *BoolFalse*, the FEP should be off. If the function returns *BoolTrue*, the FEP should be on.

Concurrency: Synchronous

### 30.5.9 getSetting()

Protected member of: **SysConfigTable**

Return Class: **void**

Arguments:

**unsigned** *item*
**unsigned short&** *value*
**Boolean&** *changed*

Documentation:

This function reads and resets the changed flag for the indicated *item*. The fetched item is returned in *value*. If the value has changed since the last time it was fetched, the function sets *changed* to *BoolTrue*, otherwise, *changed* contains *BoolFalse*.

Concurrency: Synchronous

### 30.5.10 getTableInfo()

Public member of: **SysConfigTable**

Return Class: **void**

Arguments:

**const unsigned*&** *addr*
**unsigned&** *wordcnt*

Documentation:

This function returns the address and word length of the system configuration table. Upon return, *addr* points to the start of the system configuration table, and *wordcnt* contains the number of 32-bit words in the table.

Concurrency: Synchronous

## 30.6 Class SystemConfiguration

Documentation:

This class manages the system configuration of the instrument. It is responsible for monitoring the radiation flag, maintaining the power settings to the DEA and FEP, and updating DEA register settings.

Export Control: Public

Cardinality: 1

Hierarchy:

Superclasses: **Task**

Implementation Uses:

**SysConfigTable**
**RadDevice**
**TaskMonitor**
**FepManager**
**DeaManager**
**ScienceManager**
**IntrGuard**

Public Interface:

Operations:

```
SystemConfiguration()
goTaskEntry()
reloadCcdSettings()
```

Protected Interface:

Operations:

```
checkMonitors()
enableScience()
isRadiationOn()
powerOffDea()
updateCcdSettings()
updateCntlSettings()
updateDeaPower()
updateFepPower()
```

Private Interface:

Has-A Relationships:.

**const unsigned** *pollSleep*: This contains the number of ticks the task should sleep on each iteration.

**Boolean** *radiationAssertFlag*: This flag indicates that the radiation monitor has been asserted.

Concurrency: Active

Persistence: Persistent

### 30.6.1 SystemConfiguration()

Public member of: **SystemConfiguration**

Arguments:

**unsigned** *taskid*

Documentation:

This is the constructor of the system configuration class. *taskid* is the RTX task id to use for the task. This function first initializes the parent **Task**. It then initializes its instance variables.

Concurrency: Sequential

### 30.6.2 checkMonitors()

Protected member of: **SystemConfiguration**

Return Class: **Boolean**

Documentation:

This function polls the radiation monitor and the task monitor. If the task monitor is querying the task, the function responds to the query. If the radiation monitor is off, the function returns *BoolFalse*. If the monitor is on, the function returns *BoolTrue*.

Concurrency: Synchronous

### 30.6.3 enableScience()

Protected member of: **SystemConfiguration**

Return Class: **void**

Documentation:

This function enables science activities if the radiation monitor is not active.

Concurrency: Synchronous

### 30.6.4 goTaskEntry()

Public member of: **SystemConfiguration**

Return Class: **void**

Documentation:

This is the main loop of the system configuration task. This loop sleeps for *pollSleep* timer ticks. It then processes *taskMonitor* queries, changes to the radiation monitor flag, and updates any changed power or board settings, if allowed by the radiation monitor and science.

Concurrency: Synchronous

### 30.6.5 isRadiationOn()

Protected member of: **SystemConfiguration**

Return Class: **Boolean**

Documentation:

This function polls the radiation monitor. If the monitor is asserted the function returns *BoolTrue*. If the monitor is off, the function returns *BoolFalse*.

Concurrency: Synchronous

### 30.6.6 powerOffDea()

Protected member of: **SystemConfiguration**

Return Class: **void**

Documentation:

This function shuts down power to all of the DEA's CCD-controller boards.

Concurrency: Synchronous

### 30.6.7 reloadCcdSettings()

Protected member of: **SystemConfiguration**

Return Class: **Boolean**

Arguments:

**CcdId** *ccdid*

Documentation:

This function reloads all settings to the DEA CCD controller board indicated by *ccdid*. Currently, the function always returns *BoolTrue*.

Concurrency: Synchronous

### 30.6.8 updateCcdSettings()

Protected member of: **SystemConfiguration**

Return Class: **Boolean**

Documentation:

This function updates any settings which have changed in any of the powered DEA CCD controllers. It returns *BoolTrue* if it completes, and *BoolFalse* if it is aborted by the radiation monitor.

NOTE: This function is currently unused, and may disappear.

Concurrency: Synchronous

### 30.6.9 updateCntlSettings()

Protected member of: **SystemConfiguration**

Return Class: **void**

Documentation:

This function updates any settings which have changed in the DEA Interface controller.

Concurrency: Synchronous

### 30.6.10 updateDeaPower()

Protected member of: **SystemConfiguration**

Return Class: **Boolean**

Documentation:

This function tests DEA power, first shutting down all boards which should be turned off, and then powering on boards which should be powered on. This function returns `BoolTrue` if successful, and `BoolFalse` if it was aborted by the radiation monitor.

Concurrency: Synchronous

### 30.6.11 updateFepPower()

Protected member of: **SystemConfiguration**

Return Class: **Boolean**

Documentation:

This function updates the power settings of the FEPs. It first powers down all FEPs which should be off, and then powers on and runs the default FEP program on all FEPs which should be powered on. This function currently always returns `BoolTrue`.

Concurrency: Synchronous