cdevCollection Device Requirements and Design

Walt Akers, Chip Watson, Jie Chen

Version 1.5 December 9, 1996

TJNAF - Thomas Jefferson National Accelerator Facility

DOCUMENT DATE: Table of contents generated: December 9, 1996 11:44 am

TRADEMARKS: UNIX is a registered trademark of AT&T in the USA and other countries.

VxWorks is a register trademark of Wind River Systems.

The X Window System is a trademark of Massachusetts Institute of Technology.

OSF/Motif and Motif are trademarks of Open Software Foundation, Inc.

Ultrix and DEC are registered trademarks of Digital Equipment Corporation.

HPUX is a registered trademark of Hewlett Packard.

SURA/CEBAF: The Southeastern Universities Research Association (SURA) operates the Continuous Electron Beam Accelerator Facility (CEBAF) for the United States Department of Energy under contract DE-AC05-84ER40150.

Table of Contents

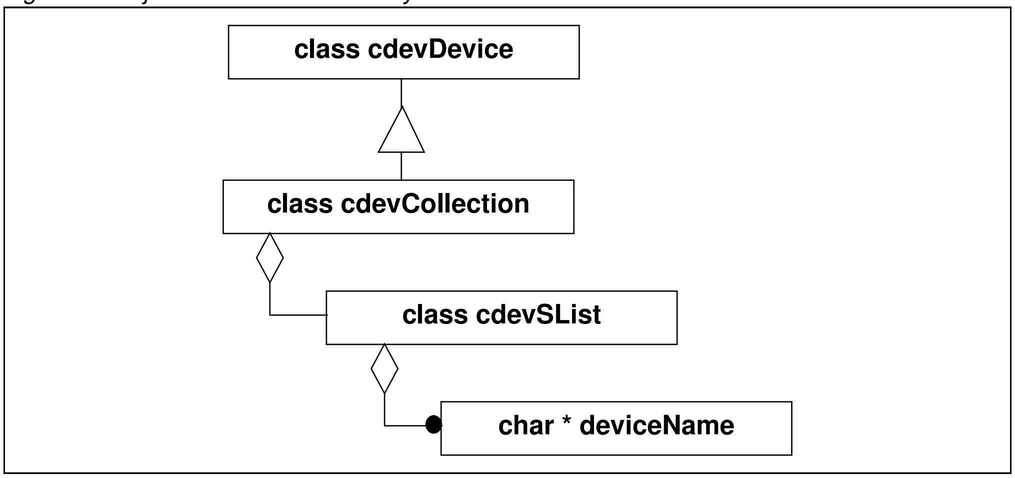
List of Figures

1. cdevCollection Class Design

Purpose of the Class

The cdevCollection class provides a mechanism for sending a message to a collection of named devices. The object inheritance hierarchy of the cdevCollection is described in the following object diagram.

Figure 1: Object Inheritance Hierarchy of the cdevCollection Class



Requirements of the Class

- A *cdevCollection* is a *cdevDevice* and is a named entity within the CDEV system of objects.
- The name of the *cdevCollection* must be unique within the CDEV system of objects.
- Only one instance of a named *cdevCollection* may exist within an executing application.
- The *cdevCollection* object contains a single linked list of device names. These device names define the collection.
- A *cdevCollection* object may be defined in the *CDEV DDL* file or may be dynamically created at run-time.
- The *CDEV DDL* definition of a *cdevCollection* object contains the names of all device names that comprise the object. However, names may be added to this list after it has been instantiated.
- Device names may be added to or removed from a *cdevCollection* object using the *add* and *remove* methods of the *cdevCollection* class.
- The component device names of a *cdevCollection* object may be added or removed until the first message has been sent to the *cdevCollection*, after which the collection of device names cannot be modified.
- A device name can only occur once within a *cdevCollection* object.

- When a device name string is added to the *cdevCollection* object using the *add* method, the method makes a copy of the string. The original string remains the property of the caller.

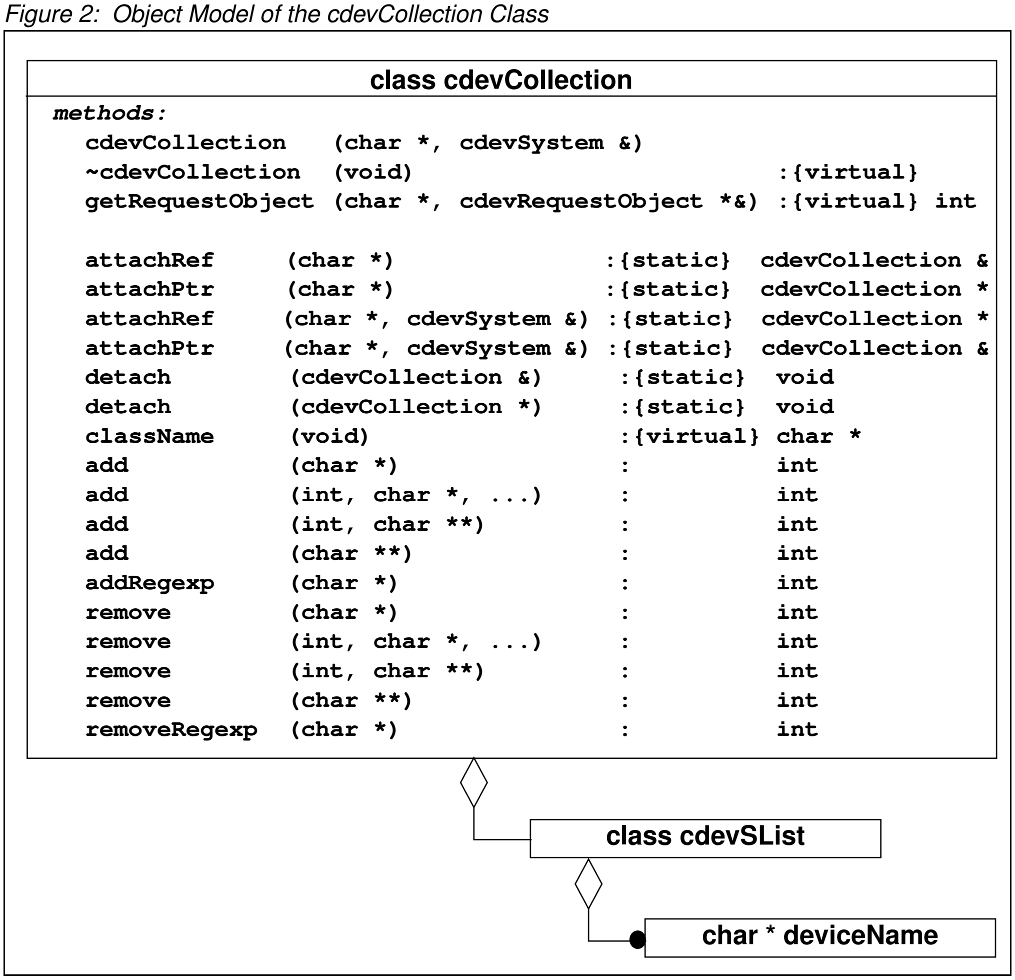
- When a device name string is removed from the *cdevCollection* object using the remove method, the string that was generated when the device name was added will be automatically deleted.

- When the *cdevCollection* object is destroyed, it is responsible for deleting any of the device name strings that it has allocated.

- The *getRequestObject* method of a *cdevCollection* object returns a *cdevCollectionRequest* object. This object inherits from (and is in the form of) a *cdevRequestObject* object.

- A *cdevCollection* object inherits its *send, sendCallback and sendNoBlock* functionality from the *cdevDevice* object. This functionality should be automatic because the *cdevDevice* object calls the *getRequestObject* method to obtain a request object for the device/message combination associated with the request.

Figure 2: Object Model of the cdevCollection Class



Method Overview of the Class

cdevCollection

*cdevCollection (char * name, cdevSystem & system);*

This is the constructor for the cdevCollection class. It has the following properties.

- This method is ***protected*** to prevent *cdevCollections* from being instantiated directly. New *cdevCollection* instances are created by using the *attachPtr* or *attachRef* method of the *cdevCollection* class.
- The name that is provided to the *cdevCollection* is the device name that CDEV will use to refer to it. The name must be unique within the CDEV system of objects and cannot be the same as the name of any regular *cdevDevice* that exists in the CDEV DDL file.
- The *cdevSystem* reference that is provided is the *cdevSystem* instance that will be used to *poll*, *pend* and *flush* the *cdevCollection* object.
- If the name provided is the name of a *cdevCollection* that has been specified in the *CDEV DDL* file, then the list of devices will be populated from the names that are specified. Otherwise, an empty *cdevCollection* instance will be created.

~cdevCollection

virtual ~cdevCollection (void);

This is the destructor for a cdevCollection object. It has the following properties.

- This method is protected to prevent the *cdevCollection* object from being destroyed by the application. This method should only be called by the *cdevSystem* object when the application is terminating.
- Because the *cdevCollection* object will normally be referred to as a *cdevDevice* object, this destructor is virtual to ensure that the ***'most senior'*** destructor is called first.
- This method will delete all device name strings that are associated with the *cdevCollection* object.

getRequestObject

*virtual int getRequestObject (char *msg,*
cdevRequestObject &req);*

This method is called in order to obtain the cdevRequestObject associated with the specified device/msg combination. This method returns CDEV_SUCCESS if the cdevRequestObject was successfully obtained. This method has the following properties.

- The *cdevRequestObject* pointer that is returned is actually a pointer to a *cdevCollectionRequest* object.
- If a *cdevCollectionRequest* already exists for the specified device/msg combination, it will be returned. Otherwise, the

method will call the *attachPtr* method of the *cdevCollectionRequest* in order to obtain a new request object.

attachRef/attachPtr

*cdevCollection & attachRef(char * name);*
*cdevCollection & attachRef(char * name, cdevSystem & sys);*
*cdevCollection * attachPtr(char * name);*
*cdevCollection * attachPtr(char * name, cdevSystem &sys);*

These methods are used to obtain a pointer or reference to a cdevCollection object. The name variable specifies the name of the collection to be attached. These method have the following properties.

- All functionality for these methods is stored in the *attachPtr (char *, cdevSystem &)* method. All other methods will call this one.

- The method will first determine if the device has already been created. If it has been created and its *className* method returns "*cdevCollection*", then the existing *cdevCollection* will be returned. If a non-collection *cdevDevice* of the same name has already been constructed, then NULL will be returned.

- If a device with the specified name has not already been created, then the method will search the *cdevDirectory* for the specified name. If the device name is in the *cdevDirectory* AND is of class "*collection*", then a new *cdevCollection* will be instantiated and returned. If a non-collection *cdevDevice* of the same name exists in the *cdevDirectory*, then NULL will be returned.

- If a device with the specified name has not been instantiated and does not exist in the *cdevDirectory*, then a new, empty *cdevCollection* will be created with the specified name.

- If NULL is returned to one of the *attachRef* methods, then a reference to a *cdevErrorCollection* object will be returned to the caller.

- The *refCount* property of the *cdevCollection* will be incremented each time a copy of the *cdevCollection* object is obtained by the application.

detach

int detach (cdevCollection & col);
*int detach (cdevCollection * col);*

This method is called to detach a pointer or reference to a cdevCollection object. It has the following properties.

- This method decrements the *refCount* property of the *cdevCollection* object.

- When the *refCount* property is decremented to 0, then the *cdevCollection* object will be deleted.

className *char * className (void);*

This method returns the name of the class; "*cdevCollection*". If the developer inherits a service specific cdevCollection, then this method should not be altered or overridden.

add *int add (char *name);*

This method will add the individual name to the list of device names that are in the cdevCollection. This method has the following properties.

- If a *cdevCollectionRequest* object has been instantiated for this *cdevCollection* device, then the operation will fail and **CDEV_INVALIDOP** will be returned.
- If the name provided is NULL or empty, then the operation will fail and **CDEV_INVALIDOP** will be returned.
- If the name specified already exists in the collection of names, then the operation will fail and **CDEV_WARNING** will be returned.
- If the name specified is not already in the list, then the method will append the device name to the end of the list and return **CDEV_SUCCESS**.

add *int add (int num, char *firstname,...);*

This method will attempt to add a list of num names to the cdevCollection. This method has the following properties.

- If a *cdevCollectionRequest* object has been instantiated for this *cdevCollection* device, then the operation will fail and **CDEV_INVALIDOP** will be returned.
- This method will call the *add (char *name)* method for each name that is specified.
- If the add method fails to add any of the device name strings to the list, then the error code that was generated by the call to *add (char * name)* will be returned.
- If an error occurs while adding a name to the list, the method will continue with the next name until the last name is reached.
- If no errors occur while adding names to the list, then **CDEV_SUCCESS** will be returned to the caller.

add *int add (int num, char **names);*

This method will attempt to add *num* device names from the *names* array to the list using the *add (char *name)* method of the *cdevCollection*. This method has the same properties as the *add (int num, char * firstname,...)* method.

add *int add (char **names);*

This method will attempt to add each device name from the NULL terminated *names* array to the list using the *add (char *name)* method of the *cdevCollection*. This method has the same properties as the *add (int num, char * firstname,...)* method.

addRegexp *int addRegexp (char *regexp);*

This method will query the system name server for the list of all names in the *cdevDirectory* that match the specified regular expression. the array of names will be added to the list using the *add (int num, char ** names)* method and the result of this operation will be returned.

remove *int remove (char *name);*

This method will remove the specified device name from list of device names that is stored in the *cdevCollection* object. This method has the following properties.

- If a *cdevCollectionRequest* object has been instantiated for this *cdevCollection* device, then the operation will fail and **CDEV_INVALIDOP** will be returned.
- If the name provided is NULL or empty, then the operation will fail and **CDEV_WARNING** will be returned.
- If the name specified does not exist in the collection of names, then the operation will fail and **CDEV_WARNING** will be returned.
- If the name specified exists in the list, then it will be removed and **CDEV_SUCCESS** will be returned.

remove *int remove (int num, char *name,...);*

This method will attempt to remove a list of num names from the *cdevCollection*. This method has the following properties.

- If a *cdevCollectionRequest* object has been instantiated for this *cdevCollection* device, then the operation will fail and **CDEV_INVALIDOP** will be returned.
- This method will call the *remove (char *name)* method for each name that is specified.

- If the remove method fails to remove any of the device name strings from the list, then the error code that was generated by the call to *remove (char * name)* will be returned.

- If an error occurs while removing a name from the list, the method will continue with the next name until the last name is reached.

- If no errors occur while removing names from the list, then **CDEV_SUCCESS** will be returned to the caller.

remove *int remove (int num, char **names);*

This method will attempt to remove *num* device names as specified in the *names* array from the list using the *remove (char *name)* method of the *cdevCollection*. This method has the same properties as the *remove (int num, char * name,...)* method.

remove *int remove (char **names);*

This method will attempt to remove each device name as specified in the NULL terminated *names* array from the list using the *remove (char *name)* method of the *cdevCollection*. This method has the same properties as the *remove (int num, char * name,...)* method.

removeRegexp *int removeRegexp (char *regexp);*

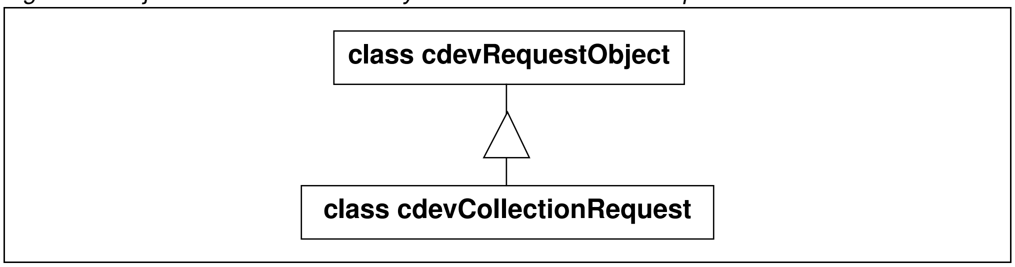
This method will query the system name server for the list of all names in the *cdevDirectory* that match the specified regular expression. the array of names will be removed from the list using the *remove (int num, char ** names)* method and the result of this operation will be returned.

2. cdevCollectionRequest Class Design

Purpose of the Class

The cdevCollectionRequest class is an abstract base class from which other cdevCollectionRequest objects a re derived. It provides a protected constructor and destructor that are used to initialize its internals, however, the primary mechanism that is used to obtain a cdevCollectionRequest object is the *attachPtr* method.

Figure 3: Object Inheritance Hierarchy of the cdevCollectionRequest Class



Requirements of the Class

- This class will provide a mechanism for the caller to obtain a pointer to a cdevCollectionRequest derived object that can be used as an interface to a collection of devices.
- The device/message combinations may fall into a variety of services.
- The derived class will provide default support for the methods send, sendCallback and sendNoBlock.
- The send and sendNoBlock methods will rely on the sendCallback method to transmit their request and receive a reply.
- Each data item that is returned by the individual devices must be scalar in nature. Non-scalar results returned by a device will be discarded.
- If the scalar data that is returned by multiple devices is not homogeneous, the data type will be promoted in such a way as to ensure that no data is lost.
- A special tagged data item, “resultCode” will be added to the cdevGlobalTagTable. This data item will hold an ordered array of the completion codes that were returned to the callback function in the *status* parameter.

Figure 4: Object Model of the cdevCollectionRequest Class

class cdevCollectionRequest

```
attributes:
  RESULT_CODE_TAG: int

methods:
  cdevCollectionRequest    (char **, int, char *, cdevSystem &)
  ~cdevCollectionRequest   (void)   :{virtual}

  attachPtr               (...)    :{static} cdevCollectionRequest
  *
  className                (void)   :{virtual} char *
```

Method Overview of the Class

constructor

*cdevCollectionRequest(char **devices, int nDevices,*
*char * msg, cdevSystem & system);*

This is the constructor for the cdevCollectionRequest class. It has the following properties.

- This method is ***protected*** to prevent the direct instantiation of new *cdevCollectionRequests*. New instances of the *cdevCollectionRequest objects are* created by using the *attachPtr* or *attachRef* method of the *cdevRequestObject* class which will call the local *attachPtr* method to create a new object if necessary.
- The constructor is called by the cdevCollection object and is provided with a list and count of devices that will be included in the collection and the message that will be sent to them.
- The *cdevSystem* reference that is provided is the *cdevSystem* instance that will be used to *poll*, *pend* and *flush* the *cdevCollectionRequest* object.

destructor

virtual ~cdevCollectionRequest (void);

This is the destructor for a cdevCollectionRequest object. It has the following properties.

- This method is protected to prevent the *cdevCollectionRequest* object from being destroyed by the application. This method should only be called by the *cdevSystem* object when the application is terminating.
- Because the *cdevCollectionRequest* object will normally be referred to as a *cdevRequestObject* object, this destructor is virtual to ensure that the ***'most senior'*** destructor is called first.

attachPtr

*cdevCollectionRequest * attachPtr*
*(cdevCollection &col, char *msg, cdevSystem &sys);*

This method is used by the cdevCollection object to obtain a new cdevCollectionRequest object.

- This method will obtain a copy of the device names from the cdevCollection object and will poll the cdevDirectory object to determine which service each of them is associated with.
- If the devices are all from a single service, this method will return a service specific collection request object.
- If the devices are from a variety of services, this method will return a cdevGrpCollectionRequest that contains the service specific collection request objects.

- Device/message combinations that are not associated with a service will be ignored.
- If none of the device/message combinations can be associated with a service, then an error message will be generated and NULL will be returned.

className

*char * className (void);*

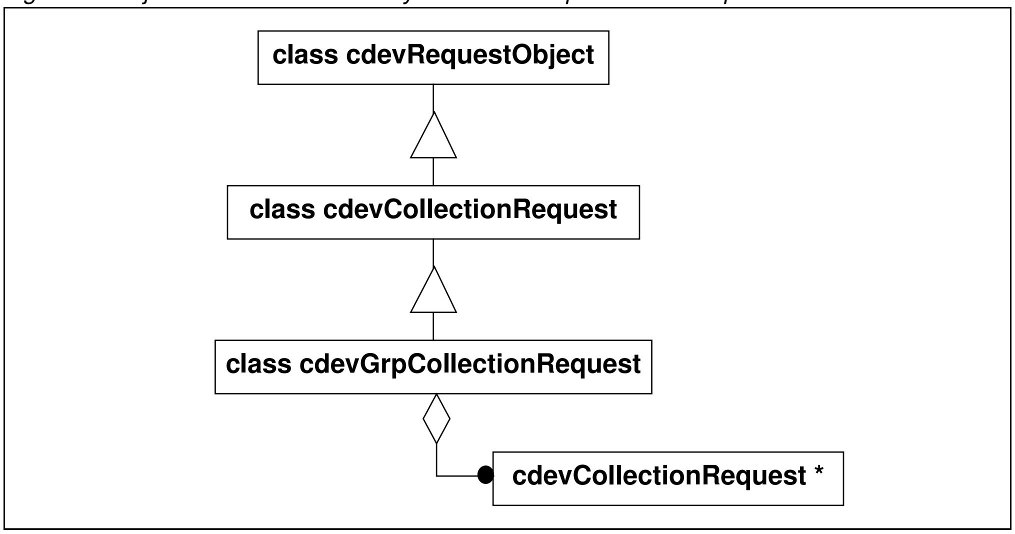
This method returns the name of the class; "*cdevCollectionRequest*". If the developer inherits a service specific cdevCollectionRequest, then this method should not be altered or overridden.

3. cdevGrpCollectionRequest Class Design

Purpose of the Class

The cdevGrpCollectionRequest class is used when the list of devices will be serviced by more than one cdevService. This class will create and maintain a list of service specific cdevCollectionRequest objects and will compile the results into a single cdevData object that will be returned to the caller.

Figure 5: Object Inheritance Hierarchy of the cdevGrpCollectionRequest Class



Requirements of the Class

- This class will break the list of devices into groups that will be associated with individual services.
- A service-specific cdevCollectionRequest will be created and maintained for each of the individual groups of devices.
- The methods send, sendNoBlock and sendCallback will be provided to allow the message to be transmitted to all of the embedded collections.
- The send and sendNoBlock methods will utilize the sendCallback method to transmit messages. This will reduce the amount of code that is required to perform message transmission.
- The class will collect all of the data from the service-specific collections into a single cdevData object that will be provided to the user. The “*resultCode*” tag will contain an array of completion codes that will indicate the completion status for each device/message request.
- The cdevGrpCollectionRequest object will maintain order information about the individual devices that exist in the various service-specific collections. Resultant data items will be placed in the correct order in the cdevData item that is returned to the caller.
- If the data arrays that are returned by multiple collections are not homogeneous, the data type will be promoted in such a way as to ensure that no data is lost.

- If a device does not have an associated cdev service, then its "*resultCode*" entry will be set to CDEV_NOT_FOUND.
- If none of the devices can be associated with a service, then any call to send, sendNoBlock or sendCallback will fail and will return CDEV_ERROR.
- If CDEV_ERROR is returned when the sendCallback method is used, then the user defined callback function will not be called.
- The send method will wait for a predefined period (5 seconds) for all of the service specific collection objects to return their results. If one or more of the service specific collections fails to respond within that period, then the send method will fail and will return CDEV_ERROR.
- When using the sendCallback method, if one or more of the service specific collections fails to respond, the user defined callback method will never be called.
- Calls to the getState and getAccess methods will poll each of the embedded collections and will return an enumerated type that reflects the most restrictive state that exists.
- Calls to the setContext method will result in the setContext method being called for each of the embedded service-specific collections.

Figure 6: Object Model of the cdevGrpCollectionRequest Class

class cdevGrpCollectionRequest

```
attributes:
  sendStatus    : struct SendStatus
  sendCheckSum  : int
  collections   : cdevCollectionRequest **
  nCollections  : int
  requestOrder  : int *
  nRequests     : int
  format        : cdevData

methods:
  cdevGrpCollectionRequest   (char **, int, char *, cdevSystem &)
  ~cdevGrpCollectionRequest  (void)          :{virtual}

  getState     (void)                        :{virtual} int
  getAccess    (void)                        :{virtual} int
  setContext   (cdevData &)                  :{virtual} int
  send         (cdevData &, cdevData &)      :{virtual} int
  send         (cdevData &, cdevData *)      :{virtual} int
  send         (cdevData *, cdevData &)      :{virtual} int
  send         (cdevData *, cdevData *)      :{virtual} int
  sendNoBlock  (cdevData &, cdevData &)      :{virtual} int
  sendNoBlock  (cdevData &, cdevData *)      :{virtual} int
  sendNoBlock  (cdevData *, cdevData &)      :{virtual} int
  sendNoBlock  (cdevData *, cdevData *)      :{virtual} int
  sendCallback (cdevData &, cdevCallback &)  :{virtual} int
  sendCallback (cdevData *, cdevCallback &)  :{virtual} int
  className    (void)                        :{virtual} char *
```

Attribute Overview of the Class

sendStatus

typedef struct {
int completionCode;
int finished;
*cdevData *data;*
} SendStatus sendStatus;

This is a structure that is provided as the void * argument to the callback function that is used for the *send* method. This structure identifies the cdevData object that is to be populated with the results and is also used to maintain a *completionCode* and the *finished* semaphore.

sendCheckSum

int sendCheckSum;

Because the send method uses the sendCallback method to transmit its requests, a callback associated with an earlier (timed-out) call to send may occur while the cdevGrpCollectionRequest object is waiting. The sendCheckSum is used by the callback method to ensure that the correct request is being processed.

collections

*cdevCollectionRequest * collections;*

This is an array of the service-specific collections. The first entry in this list is a NULL collection that is used to identify all devices that are not supported by a service.

nCollections

int nCollections;

This is the number of cdevCollectionRequests that have been allocated to the collections array.

requestOrder

*int * requestOrder;*

This is an array of integers that corresponds to each device that is in the cdevCollection. Each entry contains an integer that is the index of the service-specific cdevCollectionRequest that is associated with the individual device. This array is used to place the data items from the various devices in the correct order in the resultant cdevData item.

nRequests

int nRequests;

This is the number of entries in the requestOrder array (the number of devices in the collection).

format

cdevData format;

This is a cdevData object that has the same internal structure as the first data object retrieved from the cdevGrpCollectionRequest. This cdevData object is used to pre-format the user provided cdevData objects to the correct data types in order to reduce the time expended in reformatting the incoming data during type promotions.

Method Overview of the Class

constructor

cdevGrpCollectionRequest
*(char **devices, int nDevices, char *msg, cdevSystem& sys);*

This is the constructor for the cdevGrpCollectionRequest object. It will descend the list of devices that is provided and will identify the cdev services that are associated with them. The constructor will then create a service-specific cdevCollectionRequest object for each group of devices.

Devices that are not associated with a service will be placed on a 'dead' list and will automatically be marked with the resultCode CDEV_NOT_FOUND whenever the send, sendNoBlock or sendCallback methods are employed.

destructor

~cdevGrpCollectionRequest (void);

This is the destructor for the class. It will delete all of the service-specific cdevCollectionRequests and will free any memory that was previously allocated to the class.

getState

int getState (void);

This method calls the getState method of all of the service-specific cdevCollectionRequest objects. The method will return the most restrictive state that is encountered from any its embedded collections.

getAccess

int getAccess (void);

This method calls the getAccess method of all of the service-specific cdevCollectionRequest objects. The method will return the most restrictive state that is encountered from any of its embedded collections.

setContext

int setContext (cdevData &ctx);

This method will call the setContext method of all of the service-specific cdevCollectionRequest objects. This method will always return CDEV_SUCCESS.

send

int send (cdevData & in, cdevData & out);
*int send (cdevData * in, cdevData & out);*
*int send (cdevData & in, cdevData * out);*
*int send (cdevData * in, cdevData * out);*

The send methods are used to synchronously transmit a message to all of the devices that are embedded within the cdevGrpCollectionRequest object. The method will employ the sendCallback method to transmit the message and then will wait for a predefined period (5 seconds) for all of the collections to respond. The method will display an error and return CDEV_ERROR if the collections do not reply within the time period.

sendNoBlock

int sendNoBlock (cdevData & in, cdevData & out);
*int sendNoBlock (cdevData * in, cdevData & out);*
*int sendNoBlock (cdevData & in, cdevData * out);*
*int sendNoBlock (cdevData * in, cdevData * out);*

The sendNoBlock methods are used to asynchronously send a message to all of the devices that are embedded within the cdevGrpCollectionRequest object. The method relies on the sendCallback method to transmit the request. The caller must have established a cdevGroup prior to submitting the message in order to detect when all of the collections have responded.

sendCallback

int sendCallback (cdevData & in, cdevCallback & callback);
*int sendCallback (cdevData * in, cdevCallback & callback);*

The sendCallback methods are used to asynchronously transmit a message to all of the devices that are embedded within the cdevGrpCollectionRequest object. The user defined callback method will be called when all of the service-specific collections have responded.

If one or more of the service-specific collections fails to respond, then the user specific callback will never be executed.

A copy is made of the callback object that is provided by the user to ensure that it is not inadvertently deleted before the callback function has been called.

className

*const char *className (void) const;*

This method will return the name of the class: *“cdevGrpCollectionRequest”.*

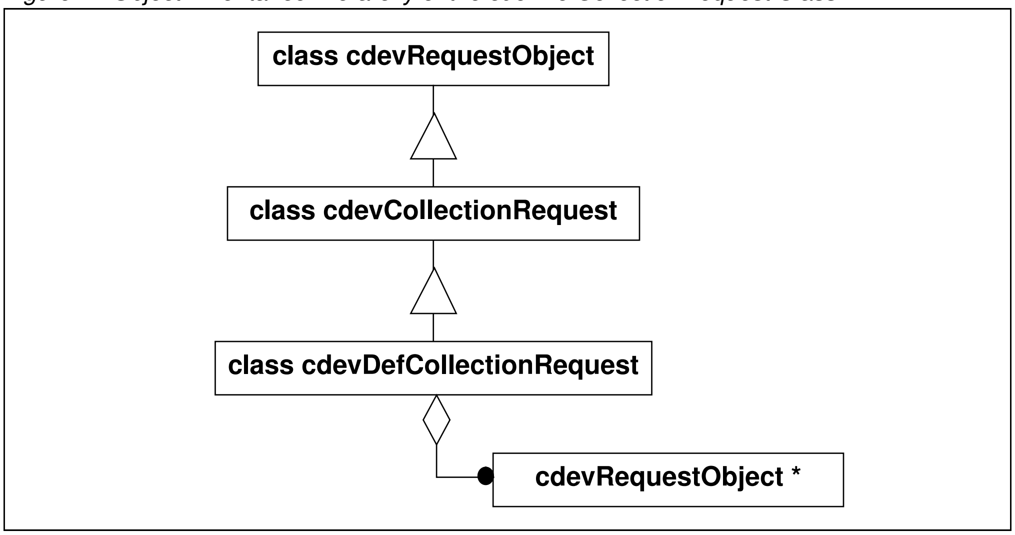
4. cdevDefCollectionRequest Class Design

Purpose of the Class

The cdevDefCollectionRequest object is the default collection request object for cdev. If the service developer does not provide a service-specific collection mechanism, then cdev will use this object to support collection request calls to the service.

This class performs collection operations by walking through the list of service-specific request objects and executing them one at a time. The results from the individual request objects will then be compiled into a single cdevData object that is returned to the caller.

Figure 7: Object Inheritance Hierarchy of the cdevDefCollectionRequest Class



Requirements of the Class

- This class will obtain a service-specific cdevRequestObject for each device that is specified.

- The methods send, sendNoBlock and sendCallback will be provided to allow the message to be transmitted to all of the embedded request objects.

- The send and sendNoBlock methods will utilize the sendCallback method to transmit messages. This will reduce the amount of code that is required to perform message transmission.

- The class will collect all of the data from the service-specific request objects into a single cdevData object that will be provided to the user. The "*resultCode*" tag will contain an array of completion codes that will indicate the completion status for each device/message request.

- The cdevDefCollectionRequest object will maintain order information about the individual devices. Resultant data items will be placed in the correct order in the cdevData item that is returned to the caller.

- If the data that is returned by the individual devices is not homogeneous, the data type will be promoted in such a way as to ensure that no data is lost.

- If CDEV_ERROR is returned when the sendCallback method is used, then the user defined callback function will not be called.
- The send method will wait for a predefined period (5 seconds) for all of the request objects to respond. If one or more of the request objects fails to respond within that period, then the send method will fail and will return CDEV_ERROR.
- When using the sendCallback method, if one or more of the request objects fails to respond, the user defined callback method will never be called.
- Calls to the getState and getAccess methods will poll each of the embedded request objects and will return an enumerated type that reflects the most restrictive state that exists.
- Calls to the setContext method will result in the setContext method being called for each of the embedded request objects.

Figure 8: Object Model of the cdevDefCollectionRequest Class

class cdevDefCollectionRequest

```
attributes:
  sendStatus    : struct SendStatus
  sendCheckSum  : int
  requests      : cdevRequestObject **
  nRequests     : int
  format        : cdevData

methods:
  cdevDefCollectionRequest   (char **, int, char *, cdevSystem &)
  ~cdevDefCollectionRequest  (void)          :{virtual}

  getState     (void)                        :{virtual} int
  getAccess    (void)                        :{virtual} int
  setContext   (cdevData &)                  :{virtual} int
  send         (cdevData &, cdevData &)      :{virtual} int
  send         (cdevData &, cdevData *)      :{virtual} int
  send         (cdevData *, cdevData &)      :{virtual} int
  send         (cdevData *, cdevData *)      :{virtual} int
  sendNoBlock  (cdevData &, cdevData &)      :{virtual} int
  sendNoBlock  (cdevData &, cdevData *)      :{virtual} int
  sendNoBlock  (cdevData *, cdevData &)      :{virtual} int
  sendNoBlock  (cdevData *, cdevData *)      :{virtual} int
  sendCallback (cdevData &, cdevCallback &)  :{virtual} int
  sendCallback (cdevData *, cdevCallback &)  :{virtual} int
  className    (void)                        :{virtual} char *
```

Attribute Overview of the Class

sendStatus

typedef struct {
int completionCode;
int finished;
*cdevData *data;*
} SendStatus sendStatus;

This is a structure that is provided as the void * argument to the callback function that is used for the *send* method. This structure identifies the cdevData object that is to be populated with the results and is also used to maintain a *completionCode* and the *finished* semaphore.

sendCheckSum

int sendCheckSum;

Because the send method uses the sendCallback method to transmit its requests, a callback associated with an earlier (timed-out) call to send may occur while the cdevDefCollectionRequest object is waiting. The sendCheckSum is used by the callback method to ensure that the correct request is being processed.

requests

*cdevRequestObject ** requests;*

This is an array of request objects that are associated with each of the devices in the collection.

nRequests

int nRequests;

This is the number of request objects in the requests array (the number of devices in the collection).

format

cdevData format;

This is a cdevData object that has the same internal structure as the first data object generated by the cdevDefCollectionRequest. This cdevData object is used to pre-format the user provided cdevData objects to the correct data types in order to reduce the time expended in reformatting the incoming data during type promotions.

Method Overview of the Class

constructor

cdevDefCollectionRequest
*(char **devices, int nDevices, char *msg, cdevSystem& sys);*

This is the constructor for the cdevDefCollectionRequest object. It will descend the list of devices that is provided and will connect to the cdevRequestObject associated with each device.

destructor

~cdevDefCollectionRequest (void);

This is the destructor for the class. It will free any memory that was previously allocated to the class.

getState *int getState (void);*

This method calls the getState method of all of the embedded request objects. The method will return the most restrictive state that is encountered.

getAccess *int getAccess (void);*

This method calls the getAccess method of all of the embedded request objects. The method will return the most restrictive state that is encountered.

setContext *int setContext (cdevData &ctx);*

This method will call the setContext method of all of the embedded request objects. This method will always return CDEV_SUCCESS.

send *int send (cdevData & in, cdevData & out);*
*int send (cdevData * in, cdevData & out);*
*int send (cdevData & in, cdevData * out);*
*int send (cdevData * in, cdevData * out);*

The send methods are used to synchronously transmit a message to all of the devices that are embedded within the cdevDefCollectionRequest object. The method will employ the sendCallback method to transmit the message and then will wait for a predefined period (5 seconds) for all of the request objects to respond. The method will display an error and return CDEV_ERROR if the request objects do not reply within the time period.

sendNoBlock *int sendNoBlock (cdevData & in, cdevData & out);*
*int sendNoBlock (cdevData * in, cdevData & out);*
*int sendNoBlock (cdevData & in, cdevData * out);*
*int sendNoBlock (cdevData * in, cdevData * out);*

The sendNoBlock methods are used to asynchronously send a message to all of the devices that are embedded within the cdevDefCollectionRequest object. The method relies on the sendCallback method to transmit the request. The caller must have established a cdevGroup prior to submitting the message in order to detect when all of the request objects have responded.

sendCallback *int sendCallback (cdevData & in, cdevCallback & callback);*
*int sendCallback (cdevData * in, cdevCallback & callback);*

The sendCallback methods are used to asynchronously transmit a message to all of the devices that are embedded within the cdevDefCollectionRequest object. The user defined callback method will be called when all of the request objects have responded.

If one or more of the request objects fails to respond, then the user specific callback will never be executed.

A copy is made of the callback object that is provided by the user to ensure that it is not inadvertently deleted before the callback function has been called.

className

*const char *className (void) const;*

This method will return the name of the class: *"cdevDefCollectionRequest".*

5. Defining cdevCollections in the CDEV DDL File

Overview

A cdevCollection device can be implemented in one of two ways; the developer can create an empty cdevCollection dynamically and add devices to it, or a collection entry can be placed in the CDEV DDL file that identifies the collection and its constituent devices. The syntax for defining a cdevCollection device in the CDEV DDL file is as follows.

Figure 9: cdevCollection Definition in the CDEV DDL File

```
/* service definitions */
...

/* class definitions */
...

/* device instances */
...

/* alias definitions */
...

collection cDevice1 :
device0,
device1,
device2
;

collection cDevice2 :
device3
device4
device5
;
```

In the example above two cdevCollections are defined. A cdevCollection definition is started by using the keyword “***collection***”, followed by a space and then the name of the collection. Note that the name of the collection must be unique and cannot be shared with any other device in the cdev system.

The device name is followed by a white space character (***which is required***), and then a colon, followed by another white space character and then the list of devices. The device names that are in the list may be separated by either white space of a ***comma*** delimiter. The list of devices is terminated by a ***semicolon***.

The current implementation of the CDEV DDL is order dependent. Because of this, the developer should place the collection definitions after the definitions of all devices and aliases.

6. Elementary Commands of the cdevCollection

Attaching to a cdevCollection

In order to use a cdevCollection, the developer must first use the attachRef or attachPtr method to obtain a pointer to the device. If the device name that is provided to the method is the name of a predefined collection device that exists in the CDEV DDL file, then it will be populated with the devices that are specified in the device list. If the device name is NOT specified in the CDEV DDL file, then a warning will be displayed and an empty cdevCollection will be created.

The code below illustrates how to obtain a reference to a cdevCollection using the attachRef method, and how to obtain a pointer to a cdevCollection using the attachPtr method. In the example, the “brandNew” cdevCollection will be created with no constituent devices because it is not defined in the CDEV DDL file. The “cDevice1” collection will be populated with the devices: “device0”, “device1”, and “device2” as defined in the CDEV DDL file.

Figure 10: Attach to a cdevCollection using attachPtr and attachRef

```
#include <cdevCollection.h>

void main ( void )
{
/***************************************************************
 * Obtain a reference to a cdevCollection device named
 * “brandNew”. Because this collection is not defined in the
 * CDEV DDL file, it will be created with no constituent
 * devices.

**************************************************************/
cdevCollection & brandNew =
cdevCollection::attachRef(“brandNew”);

/***************************************************************
 * Obtain a reference to a cdevCollection device named
 * “cDevice1”. This collection is defined in the CDEV DDL file
 * and will be populated with the devices: “device0”, “device1”,
 * and “device2”.
```

Detaching from a cdevCollection

Because only one copy of each cdevDevice or cdevCollection exists in the cdevSystem, the number of references to a device that are in use is tracked using a reference count variable. The detach method decrements that reference count. When the reference count reaches 0, the cdevSystem may delete the device.

The following illustration shows how to detach from a cdevCollection.

Figure 11: Detach from a cdevCollection using detach

```
#include <cdevCollection.h>

void main ( void )
{
cdevCollection & cDevice1 =
cdevCollection::attachRef(“cDevice1”);
cdevCollection::detach(cDevice1);
```

Adding Devices to a cdevCollection

When creating an empty cdevCollection device, the developer will have to add the constituent devices to the collection using one of the variations of the “add” method. These methods will allow the developer to add either individual devices, lists of devices, or devices from the CDEV DDL file that match a specific regular expression.

It should be noted by the developer that only cdevDevices may be added to a cdevCollection. The current design of the cdevCollection device does not support adding collections to a collection.

The variations of the add method are described in the following sections.

Adding a Single Device to a cdevCollection

When adding a single device to a collection, the *add (char * name)* method should be employed. This method will add the specified device name to the list of device names (if any) that already exists in the cdevCollection. The following source code illustrates how to add the cdevDevice “device0” to the cdevCollection “brandNew”.

Figure 12: Adding a single device to a cdevCollection

```
#include <cdevCollection.h>
void main ( void )
{
/**************************************************************
 * Attach to an undefined cdevCollection device and then add
 * the cdevDevice "device0" to it.
 **************************************************************/
cdevCollection & brandNew =
cdevCollection::attachRef("brandNew");
brandNew.add("device0");
```

Adding Multiple Devices to a cdevCollection

There are several variations of the add method that will allow you to add several devices to a cdevCollection using a single method. This method uses a variable length argument list to allow the caller to pass in an arbitrary number of device names as parameters to the call.

Figure 13: Adding multiple devices to a cdevCollection

```
#include <cdevCollection.h>

void main ( void )
{
/**************************************************************
 * Attach to the cdevCollection "brandNew" and add four device
 * names to it.
 **************************************************************/
cdevCollection & brandNew =
cdevCollection::attachRef("brandNew");
brandNew.add(4, "device0", "device1", "device2", "device3");
```

Adding an Array of Devices to a cdevCollection

This add method allows the developer to add an array of device names to the cdevCollection. In this variation of the add method, the caller must specify the number of device names that are stored in the array.

Figure 14: Adding an array of devices to a cdevCollection

```
#include <cdevCollection.h>

char * devices[] = {"device0", "device1", "device2", "device3"};

void main ( void )
{
/************************************************************
 * Attach to the cdevCollection "brandNew" and add four device
 * names to it.
 ************************************************************/
cdevCollection & brandNew =
cdevCollection::attachRef("brandNew");
brandNew.add(4, devices);
```

Adding a NULL-Terminated Array of Devices to a cdevCollection

This add method allows the developer to add an array of device names to the cdevCollection. In this variation of the add method, the last character string pointer in the array must be NULL.

Figure 15: Adding a NULL-terminated array of devices to a cdevCollection

```
#include <cdevCollection.h>

char * devices[] = {"device0", "device1", "device2", NULL};

void main ( void )
{
/************************************************************
 * Attach to the cdevCollection "brandNew" and add a NULL
 * terminated list of devices to it.
 ************************************************************/
cdevCollection & brandNew =
cdevCollection::attachRef("brandNew");
brandNew.add(devices);
```

Adding Devices to a cdevCollection Using Regular Expressions

The addRegexp method uses the regular expression parsing interface of the cdevDirectory device to obtain the names of all of the devices that match the specified expression. This list of names will then be added to the cdevCollection device. The regular expression may be in either standard format or the format used for UNIX filename expansion. See the cdevDirectory documentation for more information on regular expressions.

The following code illustrates loading all devices that match the expression "*device?*".

Figure 16: Adding a device to a cdevCollection using regular expressions

```
#include <cdevCollection.h>

void main ( void )
{
/**************************************************************
 * Attach to the cdevCollection "brandNew" and add a NULL
 * terminated list of devices to it.
 **************************************************************/
cdevCollection & brandNew =
cdevCollection::attachRef("brandNew");
brandNew.addRegexp("device?");
```

Removing a Single Device from a cdevCollection

When removing a single device from a collection, the *remove (char * name)* method should be employed. This method will remove the specified device name from the list of device names (if any) that already exists in the cdevCollection. The following source code illustrates how to remove the cdevDevice “device0” from the cdevCollection “cDevice1”.

Figure 17: Removing a single device from a cdevCollection

```
#include <cdevCollection.h>

void main ( void )
{
/**************************************************************
 * Attach to a defined cdevCollection device and then remove
 * the cdevDevice "device0" from it.
 **************************************************************/
cdevCollection & cDevice1 =
cdevCollection::attachRef("cDevice1");
cDevice1.remove("device0");
```

Removing Multiple Devices from a cdevCollection

There are several variations of the remove method that will allow you to remove several devices from a cdevCollection using a single function call. This method uses a variable length argument list to allow the caller to pass in an arbitrary number of device names as parameters to the call.

Figure 18: Removing multiple devices from a cdevCollection

```
#include <cdevCollection.h>

void main ( void )
{
/**************************************************************
 * Attach to a defined cdevCollection device and then remove
 * two cdevDevices from it.
 **************************************************************/
cdevCollection & cDevice1 =
cdevCollection::attachRef("cDevice1");
cDevice1.remove(2, "device0", "device2");
```

Removing an Array of Devices from a cdevCollection

This remove method allows the developer to remove an array of device names from the cdevCollection. In this variation of the remove method, the caller must specify the number of device names that are stored in the array.

Figure 19: Removing an array of devices from a cdevCollection

```
#include <cdevCollection.h>

char * devices[] = {"device0", "device2"};

void main ( void )
{
/*************************************************************
 * Attach to a defined cdevCollection device and then remove
 * two cdevDevices from it.
 *************************************************************/
cdevCollection & cDevice1 =
cdevCollection::attachRef("cDevice1");
cDevice1.remove(2, devices);
```

Removing a NULL-Terminated Array of Devices from a cdevCollection

This remove method allows the developer to remove an array of device names from the cdevCollection. In this variation of the remove method, the last character string pointer in the array must be NULL.

Figure 20: Removing a NULL-terminated array of devices from a cdevCollection

```
#include <cdevCollection.h>

char * devices[] = {"device0", "device2", NULL};

void main ( void )
{
/*************************************************************
 * Attach to a defined cdevCollection device and then remove
 * a NULL terminated list of devices from it.
 *************************************************************/
cdevCollection & cDevice1 =
cdevCollection::attachRef("cDevice1");
cDevice1.remove(devices);
```

Removing Devices from a cdevCollection Using Regular Expressions

The removeRegexp method uses the regular expression parsing interface of the cdevDirectory device to obtain the names of all of the devices that match the specified expression. This list of names will then be removed from the cdevCollection device. The regular expression may be in either standard format or the format used for UNIX filename expansion. See the cdevDirectory documentation for more information on regular expressions.

The following code illustrates removing all devices that match the expression "*device?*" from the cdevCollection "*cDevice1*".

Figure 21: Removing devices from a cdevCollection using regular expressions

```
#include <cdevCollection.h>

void main ( void )
{
/************************************************************
 * Attach to a defined cdevCollection device and then remove
 * all devices that match the specified regular expression.
 ************************************************************/
cdevCollection & cDevice1 =
cdevCollection::attachRef("cDevice1");
cDevice1.removeRegexp("device?");
```

Getting a List of Devices from a cdevCollection

In some cases the developer may want to obtain a list of devices that constitute a cdevCollection. This can be done by using the getList method. This method will obtain a NULL terminated list of devices that currently comprise the cdevCollection. The developer is responsible for deleting the character string array that is returned by this method, however, one should not delete the individual character strings that are stored in the array.

Figure 22: Getting a list of devices from a cdevCollection

```
#include <cdevCollection.h>

void main ( void )
{
/************************************************************
 * Attach to a cdevCollection device and then use the getList
 * method to obtain the array of names that are currently
 * stored in the cdevCollection and display them.
 *
 * Note that the list (not the list items) must be deleted by
 * the caller.
 ************************************************************/
cdevCollection & cDevice1 =
cdevCollection::attachRef("cDevice1");

char ** list = cDevice1.getList();
for(char **s = list; *s!=NULL; s++) printf("%s\n", *s);
delete list;
```

7. Restrictions of the cdevCollection Device

1. Device names can only be added to or removed from the cdevCollection device until it has been used ***once***. After a cdevCollectionRequest object has been generated from a specific cdevCollection device, the device list cannot be modified.

2. cdevCollection devices do not support the messages “monitorOn” and “monitorOff”. Only monotonic operations are currently supported by the cdevCollection device.

3. cdevCollection definitions must be the last things defined in the CDEV DDL file. This is because the constituent devices must have already been defined when the cdevCollection is defined.

4. When the data from a cdevDevice operation is used to construct the output cdevData object for a cdevCollection operation, only data that is scalar in nature will be used. If a cdevDevice returns tagged data items that are arrays in its results, those data items will not be included in the result.

5. cdevCollection objects must be comprised of regular cdevDevice objects only. Because the cdevCollection does not support cdevDevices that return arrays of data in their result, a cdevCollection can not contain another cdevCollection.

6. If a cdevCollection consists of devices from multiple services, the collection does not guarantee the order in which the cdevDevice’s requests will be serviced. Because services perform at varying speeds, a faster service may process its devices requests sooner than another service.