File System Abstraction Layer / Call Back (FSAL_CB) Interface Design Specification

Philippe DENIEL (philippe.deniel@cea.fr)

v0.1

Abstract : this document describes the FSAL_CB interface. This low level interface is to be used in nfs-ganesha to provide the daemon with feedback from the underlying file-system. This information is received as events that can be associated with callbacks. FSAL_CB module is the natural client to Cache_Inode_CB, which is to be described in another document.

Low Level FSAL_CB API

The FSAL_CB is roughly an interface based on events caught from the underlying file-system. This means that this file-system has a way to provide a user space process with such information. This may be made of many different ways, depending on the features of the file-system wrapped into the FSAL. For example:

- on top of Lustre, the changelog mechanism can be used to get every operation performed on the Lustre FS. The nfs-ganesha, via the FSAL_CB, just register to the Lustre MDS that will feed it on demand with information on updates.
- On top of a « DMAPI capable » file-system, like XFS for example, the DMAPI event bus could be used to implement a FSAL_CB event bus.

Filters are a very important in this interface. They are a way to get rid of pieces of information you do not care about. For example, a Lustre changelog will contain information on everything made on the FS, including what the nfs-ganesha daemon itself does on the underlying FS. This is not to be treated and should be filtered. On many implementations, the daemon will probably be interested only by operations that change the state of the file-system (mkdir, unlink, setattr, rename, …). The filters in FSAL_CB are the mechanism to be used to achieve this.

Events are the structures that you got from the FSAL_CB mechanism, they describe what operation was made, with the associated information. For example, if a file is deleted, provided information will include deleted entry's parent FSAL handle and deleted entry's name.

Low level structures

```
typedef struct fsal_cb_event_bus_parameter_
{
} fsal_cb_event_bus_parameter_t ;


typedef struct fsal_cb_event_bus_context_
{
} fsal_cb_event_bus_context_t ;
```

```
typedef struct fsal_cb_event_bus_filter_
{
} fsal_cb_event_bus_filter_t ;

typedef union fsal_cb_event_data__
  {
     fsal_cb_event_data_create_t create ; /* TBD */
     fsal_cb_event_data_unlink_t unlink ; /* TBD */
     fsal_cb_event_data_rename_t rename ; /* TBD */
     fsal_cb_event_data_commit_t  commit; /* TBD */
     fsal_cb_event_data_write_t write; /* TBD */
     fsal_cb_event_data_link_t link; /* TBD */
     fsal_cb_event_data_lock_t lock  ; /* TBD */
     fsal_cb_event_data_lock_t locku  ; /* TBD */
     fsal_cb_event_data_open_t open  ; /* TBD */
     fsal_cb_event_data_close_t close  ; /* TBD */
     fsal_cb_event_data_setattr_t setattr  ; /* TBD */
  } fsal_cb_event_data_t

typedef struct fsal_cb_event_
{
  unsigned int event_type,
  fsal_cb_data_t event_data ;
} fsal_cb_event_t ;
```

Low Level calls

```
fsal_status_t FSAL_CB_EB_Init ( fsal_cb_event_bus_parameter_t * pebparam, /*IN*/
                                fsal_cb_event_bus_context_t   * pcbebcontext /*OUT) ;

fsal_status_t FSAL_CB_EB_AddFilter( fsal_cb_event_bus_filter_t * pcbebfilter, /* IN */
                                    fsal_cb_event_bus_context_t * pcbebcontext /* INOUT */ ) ;

fsal_status_t FSAL_CB_EB_GetEvents( fsal_cb_event_t * pevent_array, /* OUT */
                                    fsal_count_t      event_array_nbslots, /* IN */
                                    fsal_time_t       timeout, /* IN */
                                    fsal_count_t    * peventfound /* Out */
                                    fsal_cb_event_bus_context_t * pcbebcontext ) ;
```

Implementing callbacks based on event reception

The callback mechanism is roughly based on the same logic that lead to the design of nfs-ganesha's RPC request management and which is used as well in the asynchronous metadata management. One or more thread is calling FSAL_CB_Init to acquire a fsal_cb_context which is specific to the thread (for example with different filters). If several threads are used, it is the FSAL_CB implementation's responsability to duplicate information to the threads and/or direct the event to the right thread. This can be made by a dispatcher/worker logic for example. First implementations will probably use a single thread.

The fsal_cb_event_functions_t array describes functions to be called as a given event occurs. They are associated in a structure that defines which functions is to be called for a given event. If the event is to be ignored or is not supported, then a NULL pointer value is to be used.

```
typedef struct fsal_cb_event_functions__
{
  fsal_status_t (*fsal_cb_create) (fsal_cb_event_data_create_t * pevdata ) ;
  fsal_status_t (*fsal_cb_unlink) (fsal_cb_event_data_unlink_t * pevdata ) ;
  fsal_status_t (*fsal_cb_rename) (fsal_cb_event_data_rename_t * pevdata ) ;
  fsal_status_t (*fsal_cb_commit) (fsal_cb_event_data_commit_t * pevdata ) ;
  fsal_status_t (*fsal_cb_write) (fsal_cb_event_data_write_t * pevdata ) ;
  fsal_status_t (*fsal_cb_link) (fsal_cb_event_data_link_t * pevdata ) ;
  fsal_status_t (*fsal_cb_lock) (fsal_cb_event_data_lock_t * pevdata ) ;
  fsal_status_t (*fsal_cb_locku) (fsal_cb_event_data_locku_t * pevdata ) ;
  fsal_status_t (*fsal_cb_open) (fsal_cb_event_data_open_t * pevdata ) ;
  fsal_status_t (*fsal_cb_close) (fsal_cb_event_data_close_t * pevdata ) ;
  fsal_status_t (*fsal_cb_setattr) (fsal_cb_event_data_setattr_t * pevdata ) ;
} fsal_cb_event_functions_t ;
```

A unique function is dedicated in calling these callbacks. One or more dedicated threads (as stated above) will call the FSAL_CB_callback function to perform the right callback information.

```
fsal_status_t FSAL_CB_callback( fsal_cb_event_t * pevent ) ;
```

The basic algorith for the internal thread used to manage FSAL CB will be similar to this (in the hypothesis that only one thread is used)

```
        CB_thread
        {
                /* Init step */
```

```
            Call FSAL_CB_EB_Init ;
            Call FSAL_CB_EB_AddFilter to tune the CB context ;
            while( 1 )
            {
                Call FSAL_CB_EB_GetEvents to get a pool of event to be managed.
                 While events in the obtained event array
                        {
                                Call FSAL_CB_callback ;
                                If Error found, issue error message
                        }
            }
}
```