```
/*

  <INTRO>

  Module Timeline - D0 Clock Timeline Library Package

  Description:
    This file contains all the routines that are used to manipulate a
    clock timeline. A clock timeline represents the 1113 RF buckets
    that a proton or antiproton bunch can be in while going through
    the accelerator. Each bucket represents roughly 18.8
    nanoseconds. The buckets for a timeline can be either UP (1) or
    DOWN (0).

    External routines:

      Function timelineCreate - Create a Timeline Object
      Function timelineDestroy - Release a Timeline Object
      Function timelineSet - Set a Range of Buckets in a Timeline Object
      Function timelineClear - Clear a Range of Buckets in a Timeline Object
      Function timelineGet - Get the Value of a Bucket in a Timeline Object
      Function timelineCopy - Copy a Timeline

    Internal routines:

      Function timelineMod - Compute the Modulus of a Bucket Index

  </INTRO>
  AUTHOR:

    Laura A. Paterno
    D0 Project Department
    Particle Physics Division
    Fermilab

  CREATION DATE: 01-Sep-1998

  MODIFICATIONS:

   17-Sep-1999  JFB  Corrected errors in the calculation of the modulo
                      bucket index. Introduced an internal function,
                      timelineMod, to compute the modulo value.
   25-Oct-1999  JFB  Added vxWorks headers


*/

/* VxWorks headers */

#include <vxWorks.h>
#include <logLib.h>

/* Standard Headers */

#include <string.h>

/* Local headers */

#include <bitArray.h>
#include <timeline.h>

/* The Number of RF Buckets associated with each timeline */
#define TIMELINE_SIZE 1113

  int timelineMod(int bucket);
```



```

/*+
  <AUTO EXTRACT>
    Function timelineCreate - Create a Timeline Object

    Description:
      Initializes the structures required to access and modify a D0 Clock
      Sequencer timeline.

    Inputs:
      none

    Outputs:
      none

    Returns:
      non-NULL - Success, pointer to a Timeline structure
      NULL     - Failure
-*/

Timeline timelineCreate(void)

  { /* Function timelineCreate */

  return (Timeline)bitArrayCreate(TIMELINE_SIZE);

   /* Function timelineCreate */}
```

```
/*+
  <AUTO EXTRACT>
    Function timelineDestroy - Release a Timeline Object

    Description:
      Destroys the Sequencer timeline structures .

    Inputs:
      t - a pointer to the Timeline structure to be destroyed.

    Outputs:
      none

    Returns:
      void
-*/
void timelineDestroy(Timeline t)

  { /* Function timelineDestroy */

  bitArrayDestroy((BitArrayId_t)t);

  } /* Function timelineDestroy */
```



```
/*+
  <AUTO EXTRACT>
    Function timelineSet - Set a Range of Buckets in a Timeline Object

    Description:
      Sets the timeline bits specified by the start_pos and end_pos values to
      1 (UP). All bits >= start_pos and <= end_pos will be set to 1 (UP).
      Start_pos and end_pos can be > 1113. The procedure is smart enough to
      wraparound and calculate the appropriate position (1-1113).

    Inputs:
      t         - Pointer to a Timeline structure
      start_pos - Starting bucket position to set
      end_pos   - Ending bucket position to be set

    Outputs:
      none

    Returns:
      void
-*/

void timelineSet(Timeline t,
                 int start_pos,
                 int end_pos)

  { /* Function TimelineSet */

  int ndx;

  /* Compute the start and end buckets modulo the timeline length */

  start_pos = timelineMod(start_pos);
  end_pos = timelineMod(end_pos);

  if (end_pos >= start_pos)
    {  /* The bucket range wraps */
    for (ndx = start_pos; ndx <= end_pos; ndx++)
      { bitArraySet((BitArrayId_t)t, ndx - 1); }
    }
  else
    {  /* The bucket range does not wrap */
    for (ndx = start_pos; ndx <= TIMELINE_SIZE; ndx++)
      { bitArraySet((BitArrayId_t)t, ndx - 1); }
    for (ndx = 1; ndx <= end_pos; ndx++)
      { bitArraySet((BitArrayId_t)t, ndx - 1); }
    }

  return;

  } /* Function TimelineSet */
```

```
/*+
  <AUTO EXTRACT>
    Function timelineClear - Clear a Range of Buckets in a Timeline Object

    Description:
      Sets the timeline bits specified by the start_pos and end_pos values to
      0 (DOWN). All bits >= start_pos and <= end_pos will be set to 0 (DOWN).
      Start_pos and end_pos can be > 1113. The procedure is smart enough to
      wraparound and calculate the appropriate position (1-1113).

    Inputs:
      t          - Pointer to a Timeline structure
      start_pos - Starting bucket position to clear
      end_pos    - Ending bucket position to be clear

    Outputs:
      none

    Returns:
      void
-*/

void timelineClear(Timeline t,
                   int start_pos,
                   int end_pos)

  { /* Function timelineClear */

  int ndx;

  /* Compute the start and end buckets modulo the timeline length */

  start_pos = timelineMod(start_pos);
  end_pos = timelineMod(end_pos);

  if (end_pos >= start_pos)
    {  /* The bucket range wraps */
    for (ndx = start_pos; ndx <= end_pos; ndx++)
      { bitArrayClear((BitArrayId_t)t, ndx - 1); }
    }
  else
    {  /* The bucket range does not wrap */
    for (ndx = start_pos; ndx <= TIMELINE_SIZE; ndx++)
      { bitArrayClear((BitArrayId_t)t, ndx - 1); }
    for (ndx = 1; ndx <= end_pos; ndx++)
      { bitArrayClear((BitArrayId_t)t, ndx - 1); }
    }

  return;

  } /* Function timelineClear */
```



```
/*+
  <AUTO EXTRACT>
    Function timelineGet - Get the Value of a Bucket in a Timeline Object

    Description:
      Gets the value of a bucket at position pos in a timeline. Returns 0 if
      the bit (bucket) is DOWN and 1 if the bit (bucket) is UP.

    Inputs:
      t   - Pointer to a Timeline structure
      pos - the bucket position to get the value.

    Returns:
      0 or 1 - if pos is in range 1-1113 and timeline is valid.
      99 - if pos is not in valid range
-*/

int timelineGet(Timeline t,
                int pos)

  { /* Function timelineGet */

  /* Compute bucket modulo the timeline length */

  pos = timelineMod(pos);

  if (bitArrayGet((BitArrayId_t)t, (pos - 1)) == 0)
    { return 0; }
  else
    { return 1; }

  } /* Function timelineGet */
```

```
/*+
  <AUTO EXTRACT>
    Function timelineCopy - Copy a Timeline

    Description:
      Copies one timeline to another. This is a deep copy. All bits and the
      size of the timeline are copied.

    Inputs:
      t1 - Pointer to the source Timeline

    Outputs:
      t2 - Pointer to the destination Timeline

    Returns:
      void
-*/

void timelineCopy(Timeline t1,
                  Timeline t2)

  { /* Function timelineCopy */

  int ndx;

  if ((t1 != NULL) && (t2 !=NULL))
    {
    for (ndx = 1; ndx <= TIMELINE_SIZE; ndx += 1)
      {
      if (timelineGet(t1, ndx) == 0)
        { timelineClear(t2, ndx, ndx); }
      else
        { timelineSet(t2, ndx, ndx); }
      }
    }

  return;

  } /* Function timelineCopy */
```



```
/*+
  <AUTO EXTRACT>
    Function timelineMod - Compute the Modulus of a Bucket Index

    Description:
      Compute the modulus of the bucket in the range 1..TIMELINE_SIZE.

    Inputs:
      bucket - Bucket index

    Returns:
      bucket modulo the range 1..TIMELINE_SIZE
-*/

int timelineMod(int bucket)

 { /* Function timelineMod */

 while (bucket <= 0)
   { bucket += TIMELINE_SIZE; }
 return (((bucket - 1) % TIMELINE_SIZE) + 1);

 } /* Function timelineMod */
```