iMS4-P -xxx

Image Sequence Tables
MS-SDK v1-8-2

iMS4 Rev-A, Rev-B, Rev-C

ISOMET CORP, 10342 Battleview Parkway, Manassas, VA 20109, USA.
Tel: (703) 321 8301, Fax: (703) 321 8546, e-mail: isomet@isomet.com
www.ISOMET.com
ISOMET (UK) Ltd, 18 Llantarnam Park, Cwmbran, Torfaen, NP44 3AX, UK.
Tel: +44 1633-872721, Fax: +44 1633 874678, e-mail: isomet@isomet.co.uk

iMS4 Sequence Tables

Sequences offer a convenient method to play (output) multiple Images. The order of the images in a sequence is defined by the user and this can be updated at any time the iMS4 is active. Many different images can be downloaded and remain in memory whilst DC power is applied.

Sequences are then downloaded. Each sequence points to any number of images previously loaded into iMS4 memory.

Multiple Sequences are stacked in a Queue for output on a first in, first out (FIFO) basis.

New sequences can be added (Downloaded), sequences deleted (Discarded), moved (MoveSequence) or reused (Recycle or Repeat).
The image data itself is NOT deleted or changed. There is no need to reload Image data when the sequence is modified, provided the image already exists in iMS4 memory.

Sequences may also have single frequency tone entries (ToneSequenceEntries). A Tone entry programs each channel with a constant frequency and amplitude. The compensation LUT table still applies. The input trigger would start a Tone in the queue in much the same was as an Image.
However, the clock input is ignored in Tone mode. The outputs remain static until the next trigger signal, which then increments the queue to the next Image or Tone.

NOTE
The following note applies to early edition iMS4's Pro-Controllers programmed with firmware ***before v2.1.xx***
For earlier firmware revisions, restrictions apply to the smallest image size used in a Sequence(s)

iMS4 build revision	***Minimum Size Image***	
Rev-A, Rev-B	*160 points*	*Points = Number of Image Points x Number of immediate Image Repeats*
Rev-C	*100usec duration*	*Duration = Number of Image Points x Image Clock Rate x Image Repeats* *e.g. Using a clock rate of 400KHz and no image repeats, the minimum number of image points = 40*

For images smaller than the above limits, possible work arounds include:

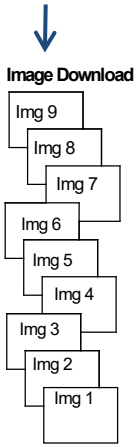
Rev-B only: *Repeat each point in the image and double the image clock rate.*
Rev-B, Rev-C: *Pad out each image with additional null points, e.g. zero amplitude points*

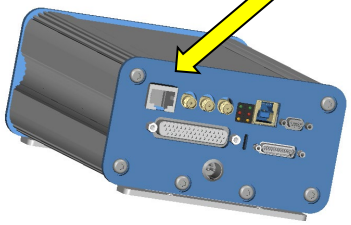
The four basic stages

1: Images

Create Images

Download Images
(Maximum 65536)





2: Sequences

Create Sequences

Sequence 1:
a: Specify termination action.
(i.e., what happens when sequence 1 completes)
b: Add required Images.

Sequence 2:
Repeat steps **a: , b:**

Sequence N:
Repeat steps **a: , b:**

Download Sequence 1 to Sequence N*.

FIFO ***ImageSequence*** queue created.

(* Up to a maximum limit N, To Be Defined)

3: Outputting

Specify ***Clock*** source for all Images in a Sequence.

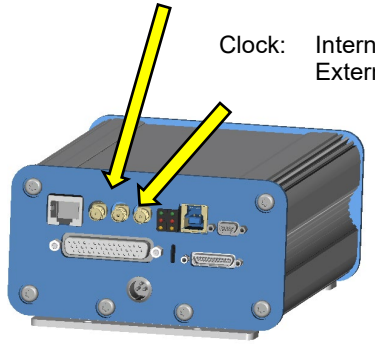
Specify ***Trigger*** source for all Images in a Sequence.

Specify the ***Trigger*** Source to start the Sequence.

Run: ***StartSequenceQueue***

Trigger: Internal (Continuous) or External J10

Clock: Internal or External J11



4: On-the-fly changes

Update the Termination action on already uploaded Sequences:

Add/Remove Sequences

Add/Remove Images

Move Sequences (in queue)

Stop, Pause or Resume

Illustrations using a queue of two sequences

Class: **SequenceManager::StartSequenceQueue**

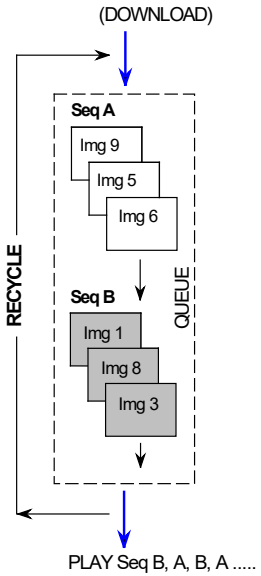
```
e.g. StartSequenceQueue(SequenceManager::SeqConfiguration(SequenceManager::PointClock::EXTERNAL, SequenceManager::ImageTrigger::EXTERNAL),
                        SequenceManager::ImageTrigger::EXTERNAL);

                    // External Clock (J11), External Image Trigger (J10) for images within sequence,
                       //External trigger (J10) for sequences within queue
```

Images are playing. The behaviour of the sequence queue depends on the termination action initially specified.

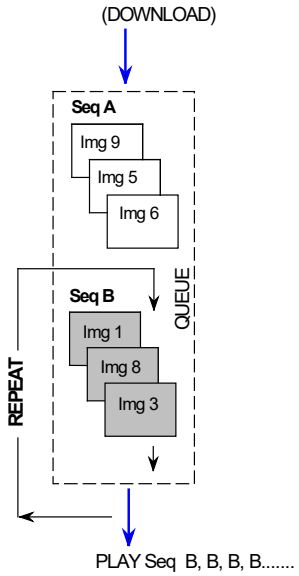
RECYCLE

```
ImageSequence seqA(SequenceTermAction::RECYCLE)
ImageSequence seqB(SequenceTermAction::RECYCLE)
```



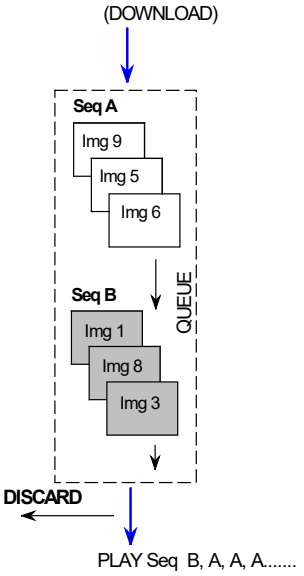
REPEAT

```
ImageSequence seqA(SequenceTermAction::RECYCLE)
ImageSequence seqB(SequenceTermAction::REPEAT)
```



DISCARD

```
ImageSequence seqA(SequenceTermAction::RECYCLE)
ImageSequence seqB(SequenceTermAction::DISCARD)
```



Additional termination actions not illustrated: ***INSERT*** and ***STOP_INSERT***.

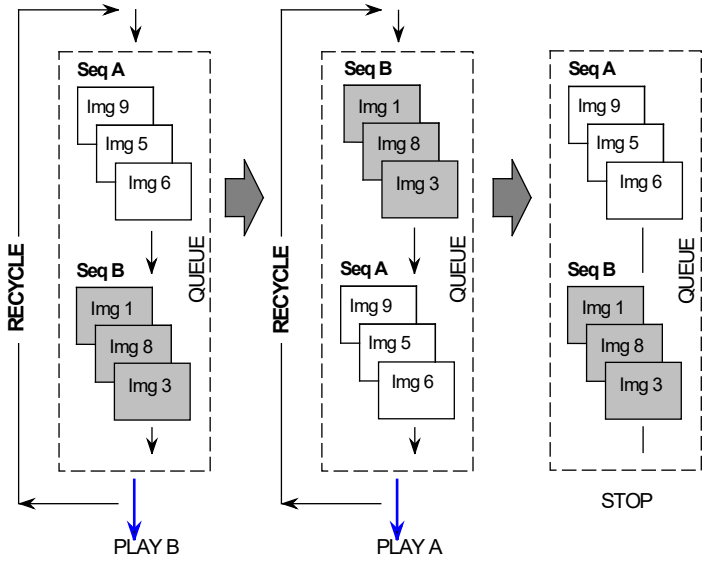
Class: **SequenceManager::UpdateTermination**

The termination action can be changed “on-the-fly” to alter the output behaviour and / or stop image play.

Stopping output.

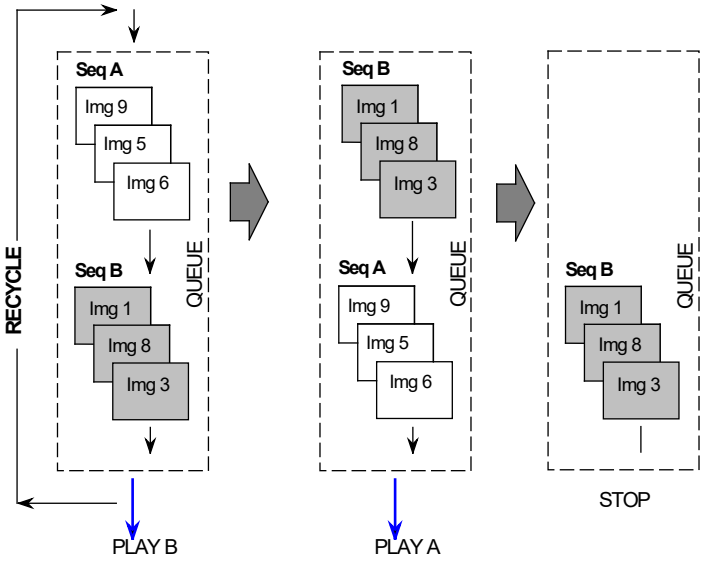
STOP_RECYCLE

```
UpdateTermination(seqA, SequenceTermAction::STOP_RECYCLE)
UpdateTermination(seqB, SequenceTermAction::RECYCLE)
```



STOP_DISCARD

```
UpdateTermination(seqA, SequenceTermAction::STOP_DISCARD)
UpdateTermination(seqB, SequenceTermAction::RECYCLE)
```

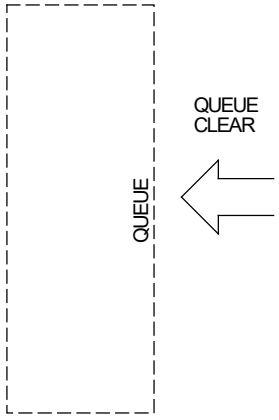


Class: **SequenceManager**

Editing Sequence queue

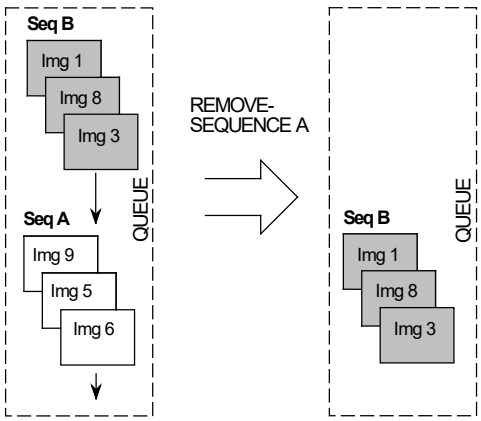
Clear entire Queue

`QueueClear()`



Removing a specific sequence

`RemoveSequence(seqA)`



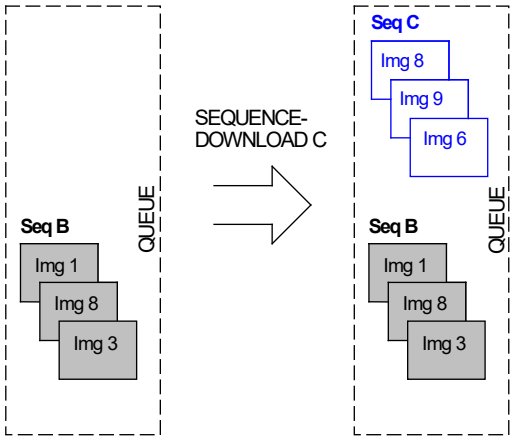
Additional actions:

Stop
Pause
Resume
MoveSequence
GetCurrentPosition
GetSequenceUUID

Class: **SequenceDownload**

Adding new sequence(s)

`SequenceDownload (SeqC)`



Example code using 2 sequences and 3 images

```
// Set Internal Clock rate for each image to be used in Sequence(s).  (Max 1000KHz or 1.0usec period)

 img1.ClockRate(kHz(ImgClk1));
// img1.ExtClockDivide(1);  // optional divider ratio if external clock selected
 img2.ClockRate(kHz(ImgClk2));
 // img2.ExtClockDivide(2);
 img3.ClockRate(kHz(ImgClk3));
 // img2.ExtClockDivide(3);



// Download images to be used in one or more Sequence(s).

 ImageDownload dl1(*myiMS, img1);
 dl1.StartDownload();

   std::this_thread::sleep_for(std::chrono::milliseconds(100)); // (avoids the C code necessary to programatically check for completion)

 ImageDownload dl2(*myiMS, img2);
 dl2.StartDownload();

   std::this_thread::sleep_for(std::chrono::milliseconds(100));

 ImageDownload dl3(*myiMS, img3);
 dl3.StartDownload();

 // Create two sequences

 ImageSequence seq1(SequenceTermAction::RECYCLE );  // Specify termination action at end of sequence (choose from one of 6 options)
 // This example applies RECYCLE termination = FIFO type behaviour; completed images added back to start of queue in continuous loop

 // Add each image to the sequence (seq1), with image options if required e.g. repeat and/or post image delay
 seq1.push_back(std::make_shared<ImageSequenceEntry>(img1, ImageRepeats::PROGRAM, 5));
 seq1.push_back(std::make_shared<ImageSequenceEntry>(img2, ImageRepeats::PROGRAM, 1));

 ImageSequence seq2(SequenceTermAction::RECYCLE);  // Specify action at end of sequence
// Add each image to the sequence (seq2), with image options if required e.g. repeat and/or post image delay
 seq2.push_back(std::make_shared<ImageSequenceEntry>(img2, ImageRepeats::PROGRAM, 4));
 seq2.push_back(std::make_shared<ImageSequenceEntry>(img3, ImageRepeats::PROGRAM, 2));
```

```
   // Download Sequences.

 SequenceManager seqMgr(*myiMS);
 seqMgr.QueueClear();              // clear any open sequences.

 SequenceDownload seqDL1(*myiMS, seq1);
 seqDL1.Download();

 std::this_thread::sleep_for(std::chrono::milliseconds(100)); // (avoids the C code necessary to programatically check for completion)

 SequenceDownload seqDL2(*myiMS, seq2);
 seqDL2.Download();

 std::this_thread::sleep_for(std::chrono::milliseconds(100));


 // Start the sequence queue.
 // Specify options for internal/external (Image) clock ,(Image)trigger options plus (Seqeunce) trigger to start the sequence


 seqMgr.StartSequenceQueue(
        SequenceManager::SeqConfiguration(
              SequenceManager::PointClock::INTERNAL, SequenceManager::ImageTrigger::CONTINUOUS),
           SequenceManager::ImageTrigger::CONTINUOUS);
           // Internal Clock (as assigned to image), Continuous Image Trigger (for images within sequence),
       //Continuous Sequence trigger (sequences within queue)

// Set iMS4 (revB) global power levels
 SignalPath Wipers(*myiMS);
 Wipers.UpdateDDSPowerLevel(DDSlvl);
 Wipers.SwitchRFAmplitudeControlSource(SignalPath::AmplitudeControl::WIPER_1);  // select ACS source = Wiper1
 Wipers.UpdateRFAmplitude(SignalPath::AmplitudeControl::WIPER_1, PsatAmp);   // set Wiper 1 value


 SystemFunc RFGate(*myiMS);          // enable RF output, connected Power Amp GATE control
 RFGate.EnableAmplifier(true);
 std::cout << "\n*** PA's Enabled ***" << std::endl;
```

```
//End output

 std::cout << "Press a key to EXIT..." << std::endl;
 std::cin.get();
 Wipers.SwitchRFAmplitudeControlSource(SignalPath::AmplitudeControl::OFF);  // select Power level to OFF

 //Update each Sequence termination. In this example with STOP_RECYCLE (sequence data is retained not discarded)
 seqMgr.UpdateTermination(seq1, SequenceTermAction::STOP_RECYCLE);
 seqMgr.UpdateTermination(seq2, SequenceTermAction::STOP_RECYCLE);

 //Commands below will remove sequences)

 seqMgr.RemoveSequence(seq1);
 seqMgr.RemoveSequence(seq2);


 RFGate.EnableAmplifier(false);
```

Example code to Stop a Sequence immediately

UpdateTermination function is used to change the behaviour of a sequence after it has completed playback.

Use a dummy ImagePlayer to Force Stop the sequence playback. For example,

```
if (SequenceManager->GetSequenceUUID(iSequenceIndex, uuidname) == true)
{
   if (SequenceManager->UpdateTermination(uuidname, SequenceTermAction::STOP_RECYCLE) == true)
   {
      // Sequence will terminate.  Send a dummy Image Player Stop command to stop playback immediately.

      ImagePlayer dummyPlayer(myiMS, Image());         // Create a dummy player object with a blank Image
      dummyPlayer.Stop(ImagePlayer::StopStyle::IMMEDIATELY);
   }
}
```

This will stop playback as soon as the Stop command is sent.
You can then flush the remaining sequences in the queue using SequenceManager->QueueClear();

Note that this procedure may leave RF output on according to the last point that was played in the sequence before the stop command was received. To clear this, use the calibration tone with zero amplitude to force the RF output off:

```
SignalPath sp(myiMS);
// Set tone to zero amplitude so no RF residue
FAP silent (0.0, 0.0, 0.0);
sp.SetCalibrationTone(silent).
sp.ClearTone();
```

Concluding Remarks

The iMS Controller will start playing the ***ImageSequenceEntry*** that exists at the front of the Sequence Queue. If the queue is empty, the call will have no effect, but may still return true. Use the ***QueueCount()*** function if it is necessary to check for queue contents prior to playback. Image playback will continue through each ***ImageSequenceEntry*** and ***ImageSequence*** in turn until it either encounters an ***ImageSequence*** with a ***'STOP_*' termination*** action or the queue becomes empty.

The queue behaviour may be controlled by the ***SeqConfiguration*** instruction, which specifies whether the Sequence is clocked by the internal NCO oscillator or an external clock and what method is used to propagate the start of the next ***ImageSequenceEntry*** in the list.

The *start_trig* parameter may be used to control how the Sequence playback begins. If set to ***CONTINUOUS*** (or not specified) Sequence playback will start immediately. If set to ***EXTERNAL***, Sequence playback will start when an External trigger is detected. If set to ***HOST***, Sequence playback will begin when a software trigger is sent (Using ***SendHostTrigger()***).

If "on-the-fly" changes are planned, the user code should include the ***SequenceEvents*** class. The software can employ this class to subscribe to the SEQUENCE_START and SEQUENCE_FINISHED events. These events indicate where a sequence is in the queue and allow the user to make "on-the-fly" changes that are synchronised with queue state.